PRATIQUE DE L'INFORMATIQUE

LES EXTENSIONS AU COBOL A.N.S.

AVEC EXERCICES ET CORRIGÉS

Ch. BONNIN

sylvine bouvry - j.p. daral

EYROLLES

LES EXTENSIONS AU COBOL A. N. S.

(avec exercices et corrigés)

CHEZ LE MÊME ÉDITEUR

Du même auteur

BONNIN	- *Le COBOL ANS avec exercices et corrigés.*
–	- *Exercices pratiques de programmation en COBOL ANS 74.*
–	- *COBOL ANS 74 (série Mémentos Eyrolles).*
–	- *Les techniques avancées de programmation PL/1.*
–	- *Pratique du PL/1.*

Autres ouvrages

STROHMEIER	- *COBOL 74. Approche systématique.*
JENSEN, WIRTH	- *Pascal. Manuel de l'utilisateur.*
HERNANDEZ	- *Pascal par l'exemple.*
CLOCKSIN	- *Programmer en Prolog.*
GNOSIS	- *Apprendre LISP.*
PETREMAN, ROUSSEAU	- *Tours de Forth.*
DELANNOY	- *Apprendre à programmer en BASIC.*
HENRIC-COLL	- *Le Forth en douceur.*

PRATIQUE DE L'INFORMATIQUE

LES EXTENSIONS AU COBOL A. N. S.

(avec exercices et corrigés)

PAR

Christian BONNIN
Ingénieur Informaticien I. D. N.
Licencié ès Sciences
Chef d'Applications Informatiques à la Compagnie IBM France

HUITIEME EDITION
Nouveau tirage

EYROLLES
61, boulevard Saint-Germain – 75005 PARIS
1986

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service «Presse», Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'article 40). »

« Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. »

© Éditions Eyrolles 1982

TABLE DES MATIÈRES

AVERTISSEMENT 1

CHAPITRE PREMIER. — **L'organisation directe** 3

Principes généraux 3
Format général d'un programme 4
Les instructions COBOL 6
Création séquentielle de fichier — Exemple 11
Création de fichier en accès sélectif — Exemple. 18
Mise à jour en accès sélectif — Exemple. 20

CHAPITRE II. — **L'organisation relative** 23

Principes généraux 23
Format général d'un programme 24
Les instructions COBOL 25
Création de fichier — Exemple 28
Mise à jour en accès sélectif — Exemple. 30

CHAPITRE III. — **L'organisation séquentiel indexé** 33

Principes généraux 33
Format général d'un programme 38
Les instructions COBOL 39
Création de fichier — Exemple 45
Mise à jour en accès sélectif — Exemple. 47
Exercice n° 1 48

CHAPITRE IV. — **VSAM** 53

Le langage AMS 53
Structure logique des fichiers VSAM 54
Fichier Index et CLUSTER 55
Confidentialité — Sécurité d'accès 56
Création d'un catalogue 56
Création de fichiers VSAM 60
Création d'un CLUSTER 62
Autres commandes AMS (ALTER-DELETE-REPRØ-LISTCAT-VERIFY) 65
Gestion des erreurs — La STATUS KEY 70

CHAPITRE V. — **L'organisation consécutive VSAM-ESDS** 73

Principes Généraux 73
Format Général d'un programme ESDS 74
Les instructions COBOL 75
Exemple de programme ESDS 78

CHAPITRE VI. — **L'organisation Indexée VSAM-KSDS** 80

Principes Généraux. 80
Format Général d'un programme KSDS 82
Les instructions COBOL 83
Exemple de programmes VSAM . Création 87
. Édition 90
. Mise-à-jour 91

CHAPITRE VII. — **L'organisation Relative RRDS** 94

Principes Généraux 94
Format Général d'un programme RRDS 95
Les instructions COBOL 96
Exemples de programmes . Création 101
. Mise-à-jour 103

CHAPITRE VIII. — **Les Index Secondaires** 106

Principes Généraux. 106
. Le cluster Index Secondaire 106
. La constitution d'un Index Secondaire. 107
. Les Index SET-UPGRADE ET NOUPGRADE 108
. La voie d'accès PATH 108
. Index uniques et Index multiples 109
. Traitements autorisés sur un fichier de base KSDS . . . 109
Les instructions du langage AMS 110
. Définition d'un index secondaire. 110
. La construction d'un fichier Index Secondaire 112
. La définition d'une voie d'accès PATH 113
Format Général d'un programme accédant à des Index Secondaires. 115
Les instructions COBOL 116
Exemple de programmation avec Index Secondaire unique. . . 118
. Création de l'Index Secondaire unique. 118
. Mise-à-jour de fichier indexé KSDS via un Index Secondaire unique 119
Exemple de programmation avec Index Secondaire multiple . . 122
. Création d'un cluster Index Secondaire multiple 122
. Édition de fichier indexé KSDS via un Index Secondaire multiple 122

CHAPITRE IX. — **La gestion des tables** 126

Principes généraux des Index 126
Les verbes de gestion des tables 128
. ASCENDING & DESCENDING KEY 128
. INDEXED BY. 129
. SET 130
. USAGE INDEX 131
. SEARCH 132
Exercice n° 2 137

CHAPITRE X. — **Le tri COBOL** 138

Principes généraux 138
Format général d'un programme. 140

Les instructions COBOL 141
. ENVIRØNMENT DIVISIØN
. DATA DIVISIØN
. PRØCEDURE DIVISIØN
Les compteurs spéciaux du tri 145
Exemple de programme de tri 147
Le verbe MERGE 149
Exercice n° 3 149

CHAPITRE XI. — **Le REPORT WRITER — Éditeur COBOL** . . . 150

Principes généraux 150
Format général d'un programme. 151
ENVIRØNMENT DIVISIØN 152
FILE SECTIØN 152
REPØRT SECTIØN 153
Les descriptions de groupes d'édition 156
Options permises ou non suivant les groupes d'édition 162
Les compteurs spéciaux 163
La PRØCEDURE DIVISIØN 164
Les sections déclaratives 166
Exemple de programme 167
Exercice n° 4. 172

CHAPITRE XII. — **La segmentation des programmes** 175
En quoi consiste la segmentation. 175
Les segments de procédure 176
Rédaction d'un programme segmenté 177
Répartition des segments 178
Points particuliers — Restrictions. 180

CORRIGÉS DES EXERCICES. 181

INDEX ALPHABÉTIQUE. 201

AVERTISSEMENT

Le langage COBOL (Common Organization Business Oriented Language) fut conçu en 1957, à la demande du Gouvernement des États-Unis qui commençait à rencontrer de sérieuses difficultés d'harmonisation des méthodes de gestion par suite de la diversité des matériels et langages utilisés au sein de ses diverses administrations.

L'étude, développée par un comité d'utilisateurs du Gouvernement américain et de constructeurs d'ordinateurs, aboutit en 1959 au système COBOL, langage basé sur l'utilisation de la langue anglaise et structuré en DIVISIONS, SECTIONS, paragraphes et phrases composées de verbes, noms, ponctuation et opérateurs (+, −, ...).

Le langage fut ensuite revu en 1961, 1965, 1968 et tout récemment en 1974 furent éditées les règles de l'U. S. A. standard COBOL, plus couramment appelé COBOL A. N. S. (American National Standard), par la CODASYL (Conference On Data System Language) et approuvées par l'ANSI (American National Standards Institute).

Intentionnellement, cet ouvrage est orienté vers la programmation des ordinateurs IBM, mais pour les lecteurs qui désireraient une stricte compatibilité, les extensions apportées par ce constructeur au COBOL A. N. S. sont distinguées par un filet vertical tremblé en marge du texte et par des zones grisées dans les expressions COBOL.

Le présent ouvrage étant destiné à enseigner la rédaction des programmes COBOL les plus évolués comprenant ACCES SELECTIF, TRI-COBOL, GESTION DES TABLES, EDITEUR COBOL et SEGMENTATION, nous avertissons le lecteur qu'il suppose la connaissance des spécifications COBOL d'usage courant détaillées dans l'ouvrage précédent « LE COBOL A. N. S. », du même auteur.

NOTATIONS UTILISÉES

Les notations utilisées pour décrire les expressions COBOL sont les expressions couramment admises, c'est-à-dire :

. Les mots réservés sont imprimés en lettres majuscules MØVE et les mots obligatoires dans une expression sont soulignés ADD.

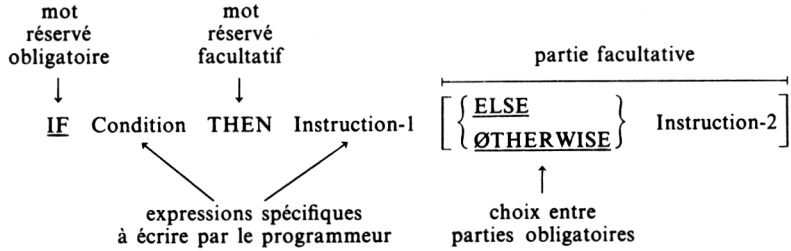
REMARQUE : Pour de nombreux mots réservés, il existe une forme pluriel possible. La forme pluriel n'ayant aucun effet sur le programme et l'usage prouvant que les programmeurs l'abandonnent très rapidement, nous n'en tiendrons pas compte dans cet ouvrage.

. Les noms symboliques ou expressions spécifiques à écrire par le programmeur sont imprimés en caractères minuscules.

. Les expressions facultatives sont imprimées entre crochets [ ]. Lorsque l'on peut choisir entre plusieurs expressions facultatives, elles sont regroupées entre crochets.

. Lorsque l'on peut choisir entre plusieurs expressions, mais que, au moins, l'une d'elles est obligatoire, elles sont regroupées entre accolades { }.

Exemple de notations :



REMARQUE : Théoriquement les normes officielles de programmation énoncent que la lettre *O* doit être écrite O et le zéro doit être barré Ø. En pratique, les programmeurs utilisent généralement la notation inverse lettre Ø et chiffre O ; c'est cette notation que nous adoptons.

LES MÉTHODES D'ACCÈS SÉLECTIVES

Outre le traitement des fichiers séquentiels purs, le COBOL A. N. S. autorise l'utilisation de fichiers à accès sélectifs organisations

DIRECTE
RELATIVE
SEQUENTIEL INDEXE

ainsi que les nouvelles organisations VSAM :

CONSECUTIVE ESDS
INDEXEE KSDS

Ces organisations supposent naturellement un support physique approprié (disque, tambour, etc.) c'est-à-dire une unité à accès sélectif adressable par un numéro de piste physique.

Pour plus de simplicité, nous allons étudier successivement chacune de ces méthodes d'accès.

CHAPITRE PREMIER

L'ORGANISATION DIRECTE

PRINCIPES GÉNÉRAUX

L'organisation directe consiste à désigner un enregistrement logique de fichier sur mémoire à accès sélectif par le numéro de la piste qui le contient puis à rechercher séquentiellement sur la piste l'indicatif qui caractérise l'enregistrement.

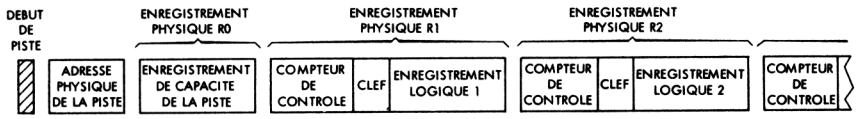
Un enregistrement logique est donc défini par | numéro de piste
clef d'enregistrement.

Cette méthode d'accès est très performante, mais suppose qu'on puisse établir aisément et sans ambiguïté une relation entre INDICATIF d'ENREGISTREMENT et | numéro de piste
clef d'enregistrement.

Dans un fichier en organisation directe les pistes sont numérotées de zéro pour la première à N – 1 pour la Nième et seuls les enregistrements NON BLOQUÉS sont autorisés quels que soient les formats

F fixe
V variable
U indéfini
S spanned

Une piste physique est organisée de la manière suivante :



Enregistrement physique R0 : Cet enregistrement n'est utilisé qu'en dehors du format fixe F pour désigner le nombre d'octets disponibles après le dernier enregistrement physique sur la piste.

Enregistrements physiques R1, R2, R3... : Chacun des enregistrements est composé de trois parties :

— un compteur de contrôle réservé au système d'exploitation,

— une clef (1 à 255 octets) uniquement accessible au système d'exploitation mais qu'il sera nécessaire de prévoir dans la dimension des blocs physiques sur disque,

— l'enregistrement logique proprement dit.

L'argument de positionnement ou recherche d'un enregistrement logique sera fourni à COBOL dans une zone de donnée ACTUAL KEY composée de la manière suivante :

— numéro de piste = 4 octets binaires PICTURE S9(5) pouvant atteindre la valeur maximum 65 535 puis,

— la clef de recherche comportant de 1 à 255 octets.

Cette zone de donnée doit être décrite en DATA DIVISIØN FILE SECTIØN, WØRKING-STØRAGE SECTIØN ou LINKAGE SECTIØN.

Par exemple :

```
A       B

01      CLEF.
          02  PISTE  PICTURE S9(5) CØMP VALUE ZERØ.
          02  INDIC PICTURE X(5).
```

Nous attirons particulièrement l'attention sur le fait que COBOL ne compare pas la clef de recherche à une zone particulière de l'enregistrement logique mais à la clef qui précède l'enregistrement logique sur la piste. La clef de recherche résulte en effet généralement d'un calcul plus ou moins compliqué à partir de l'indicatif d'enregistrement.

Une confusion est bien souvent faite avec le Séquentiel-Indexé où la clef est présente dans l'enregistrement logique.

FORMAT GÉNÉRAL D'UN PROGRAMME

Le format général d'un programme en organisation directe est le suivant :

A B

IDENTIFICATIØN DIVISIØN.

ENVIRØNMENT DIVISIØN.

CØNFIGURATIØN SECTIØN.

INPUT-ØUTPUT SECTIØN.

FILE-CØNTRØL.

SELECT Nom-de-Fichier
ASSIGN TØ Nom-de-Fichier-Système

ACCESS MØDE IS { SEQUENTIAL / RANDØM }

ACTUAL KEY IS Nom-Donnée-1

[TRACK-LIMIT IS entier [TRACK]].

[I-Ø-CØNTRØL.
APPLY RECØRD-ØVERFLØW ØN Nom-de-Fichier.]

DATA DIVISIØN.

FILE SECTIØN.

FD Nom-de-Fichier
{ Description du Fichier.

WØRKING-STØRAGE SECTIØN.

Description de la zone Nom-Donnée-1 affectée à ACTUAL KEY

PRØCEDURE DIVISIØN.

ØPEN { INPUT / ØUTPUT / I-Ø } Nom-de-Fichier

[SEEK Nom-de-Fichier RECØRD]

[READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
{ AT END / INVALID KEY } Ordre Impératif]

[{ WRITE / REWRITE } Nom d'Enregistrement [FRØM Nom-Donnée]
INVALID KEY Ordre Impératif]

CLØSE Nom-de-Fichier [UNIT] [WITH LØCK].

LES INSTRUCTIONS COBOL

SELECT Nom-de-Fichier de Data Division
ASSIGN TØ Nom-de-Fichier-Système

Rappelons que cette clause sert à établir une liaison entre le nom-de-fichier désigné dans la Data Division et le nom de fichier désigné dans les cartes de contrôle JCL du système d'exploitation.

Dans l'organisation directe, on écrira le Nom de Fichier Système sous la forme :

DA – {D / W} – [Nom de Fichier Symbolique de JCL]

Exemple :

SELECT FISTØCK ASSIGN TØ DA – D – FISTØCK

En principe W est utilisé lorsque l'ordre REWRITE est utilisé.

REMARQUE : En fait, D ou W peuvent être indifféremment utilisés mais le programmeur doit bien saisir les subtilités qui les différencient pour les fichiers ouverts en I-Ø (lecture-écriture).

. *Si D a été spécifié :* . l'ordre WRITE réécrit l'enregistrement à sa place initiale s'il vient d'y avoir une lecture READ pour ce même enregistrement.

. L'ordre WRITE ajoute un nouvel enregistrement en double si le READ précédent ne s'adressait pas à cet enregistrement.

. *Si W a été spécifié :* . REWRITE cherche à réécrire l'enregistrement à la bonne place.

. WRITE ajoute un nouvel enregistrement en double.

En système d'exploitation DOS, le Nom de Fichier Système a un format sensiblement différent :

SYSnnn – DA – [Numéro de type d'unité] – { {A / U} / {D / W} }

Exemple :

SELECT FICHIER ASSIGN TØ SYS015-DA-2314-A.

SYSnnn correspond au Nom Symbolique de fichier des cartes de contrôle JCL.

Numéro de type d'unité : 2311, 2314, 3330 ...

A et U sont à écrire lorsque un numéro relatif de piste par rapport au début du fichier est utilisé. Dans ce cas, le numéro de piste dans l'ACTUAL KEY doit être binaire sur quatre octets, PICTURE S9(8).

Exemple :

```
        |B
01      |ACTUALCLE.
        |02  RELATIVEPISTE PIC S9(8) CØMP.
        |02  CLEF PIC X(8).
```

D et W sont à écrire lorsque une adresse réelle de piste est utilisée sous la forme des 8 octets MBBCCHHR.

M = Numéro de module est une quantité binaire sur 1 octet, ce qui impose une petite astuce de programmation pour éliminer l'octet de gauche.

BB = Numéro d'unité dans le module est une quantité binaire sur 2 octets.

CC = Numéro de cylindre dans l'unité est une quantité binaire sur 2 octets.

HH = Numéro de piste dans le cylindre est une quantité binaire sur 2 octets.

R = Doit être égal à Low-Value (1 octet).

Exemple :

```
A       |B
01      |CHARGECLE.
        |02  M    PIC  S99   CØMP.
        |02  BB   PIC  S99   CØMP.
        |02  CC   PIC  S999  CØMP.
        |02  HH   PIC  S999  CØMP.
        |02  R    PIC  X   VALUE  LØW-VALUE.
        |02  CLEF  PIC  X(8).
01      |VRAICLE  REDEFINES  CHARGECLE.
        |02  FILLER  PIC  X.
        |02  ACTUALCLE  PIC  X(16).
```

U et W sont utilisés lorsque l'instruction REWRITE est employée.

ACCESS MØDE IS { SEQUENTIAL / RANDØM }

Cette clause sert à indiquer au système d'Exploitation quelle méthode d'accès doit être utilisée : SEQUENTIAL pour BSAM et RANDØM pour BDAM.

Plus simplement, disons que SEQUENTIAL s'adresse à la création d'un fichier en organisation Directe à partir d'un fichier séquentiel ordonné ou permet une lecture séquentielle d'un fichier en organisation Directe.

RANDØM est utilisé dans tous les cas de lecture ou écriture en accès aléatoire.

ACTUAL KEY IS Nom-Donnée-1

Comme nous l'avons vu plus haut ACTUAL KEY est une clause obligatoire.

[TRACK-LIMIT IS entier [TRACK]]

Cette clause n'est utilisée que dans le cas de création de fichiers en organisation Directe pour réserver et préformater des pistes d'enregistrements dans le dessein d'extensions futures du fichier.

Supposons par exemple un fichier à créer comportant 1 000 enregistrements logiques, à raison de dix par piste on occupe ainsi 100 pistes.

Si TRACK-LIMIT a été omis, il sera impossible de créer de nouveaux enregistrements ; par contre, si TRACK-LIMIT 200 a été spécifié, un espace supplémentaire de 100 pistes sera automatiquement préformaté par COBOL pour une extension possible du fichier.

En format fixe F, on obtient un véritable formatage des enregistrements avec un octet High-Value (FF hexadécimal) en première position de la clef d'enregistrement pour les distinguer.

En format variable V, indéfini U ou Spanned, il y a mise à jour de l'enregistrement de capacité Ro indiquant la place disponible sur la piste.

REMARQUE IMPORTANTE : Même si suffisamment de place a été allouée sur disque pour des ajouts ultérieurs, ces ajouts ne seront possibles qu'à condition que les pistes aient été préformatées par TRACK-LIMIT.

On se rappellera aussi que, conformément à la règle générale, l'allocation primaire de piste demandée dans les cartes de contrôle J C L par SPACE = ---- doit obligatoirement pouvoir être supportée d'un seul tenant.

[APPLY RECØRD-ØVERFLØW ØN Nom-de-Fichier.]

Cette clause n'est utilisée que pour le format fixe F et sert à éviter une perte de place en fin de piste. S'il ne reste pas suffisamment de place pour mettre un enregistrement entier, l'enregistrement est alors écrit à cheval sur la fin de piste et le début de la suivante.

$$\text{ØPEN} \left\{ \begin{array}{l} \text{INPUT} \\ \text{ØUTPUT} \\ \text{I-Ø} \end{array} \right\} \text{Nom-de-Fichier}$$

Comme en traitement séquentiel tout fichier en organisation directe doit être ouvert en INPUT avant le premier READ de lecture

en ØUTPUT avant le premier WRITE d'écriture
en I-Ø s'il s'agit d'un fichier en lecture et écriture utilisant les verbes READ, WRITE ou REWRITE.

SEEK Nom-de-Fichier RECØRD

L'accès aléatoire (RANDØM) à un enregistrement sur disque exigeant un délai démesurément long (plusieurs dizaines de millisecondes) par rapport aux vitesses de traitement de l'unité centrale (comptées en microsecondes), la clause SEEK est destinée à améliorer les performances d'un programme en profitant des simultanéités de traitement entre périphériques et unité centrale. Pour cela, le programmeur bâtira l'ACTUAL KEY de recherche et lancera dès que possible un ordre SEEK de positionnement de la tête de lecture-écriture sur la piste correspondant à l'enregistrement puis effectuera simultanément les traitements en unité-centrale nécessaires avant l'écriture ou la lecture effective.
Cette clause n'aura bien sûr aucun effet en lecture ou écriture séquentielle d'un fichier organisation directe car le bras d'accès est naturellement positionné pour l'enregistrement suivant.

READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
{ AT END / INVALID KEY } Ordre Impératif.

Lorsqu'un fichier en organisation directe est lu séquentiellement, l'ordre READ s'écrit simplement avec l'option AT END Ordre Impératif.
En traitement aléatoire (RANDØM) il faut utiliser l'option INVALID KEY Ordre Impératif qui provoque un débranchement lorsque l'enregistrement correspondant à l'ACTUAL KEY n'a pu être trouvé.

REMARQUE 1 : Éventuellement, il est possible de faire poursuivre la recherche sur les pistes suivantes en utilisant les paramètres ØPTCD et LIMCT des cartes de contrôle du système d'exploitation.

Par exemple, DCB = (ØPTCD = E, LIMCT = 10) impose au système d'exploitation de poursuivre la recherche sur les 10 pistes suivantes avant de déclencher une condition INVALID KEY.

En système DOS, il faut utiliser APPLY EXTENDED-SEARCH ØN Nom-de-fichier pour poursuivre la recherche sur l'ensemble du cylindre.

REMARQUE 2 : L'absence de l'option INVALID KEY est autorisée en COBOL-IBM mais peut provoquer des résultats imprévisibles en cas d'erreur. Il est donc préférable de considérer cette option obligatoire.

{ WRITE / REWRITE } Nom-Enregistrement [FRØM Nom de zone]

INVALID KEY Ordre Impératif

L'option REWRITE ne peut être utilisée que pour les fichiers en lecture-écriture aléatoire (I-Ø et RANDØM).

L'option WRITE devrait donc être réservée aux autres cas, c'est-à-dire fichiers en écriture séquentielle (ØUTPUT SEQUENTIAL) ou écriture aléatoire (ØUTPUT RANDØM) ; en fait WRITE peut aussi être utilisé à la place de REWRITE, mais les réactions de WRITE et REWRITE diffèrent quelque peu comme nous l'avons vu précédemment dans la clause SELECT.

La clause INVALID KEY est obligatoire pour prévenir les impossibilités d'écrire un enregistrement dans les cas suivants :

a) En création séquentielle (ØUTPUT SEQUENTIAL) lorsque toute la place allouée au fichier a été utilisée et qu'il ne reste plus de place disponible.

b) En accès aléatoire (RANDØM) lorsque le numéro de piste indiqué dans l'ACTUAL KEY est hors des limites du fichier.

c) En accès aléatoire (RANDØM) lorsque la place nécessaire pour écrire l'enregistrement n'est pas disponible sur la piste indiquée dans l'ACTUAL KEY.

REMARQUE : Il existe aussi une possibilité d'écrire un enregistrement sur une piste suivante en utilisant les paramètres ØPTCD et LIMCT des cartes de contrôle du système d'exploitation.

Par exemple DCB = (ØPTCD = E, LIMCT = 5) impose au système d'exploitation de rechercher la place nécessaire sur les cinq pistes suivantes avant de déclencher une éventuelle condition INVALID KEY.

Mais dans ce cas, le programmeur devra nécessairement utiliser cette même option dans tous les programmes qui utilisent ce fichier.

d) En format fixe F, on pourrait désirer supprimer des enregistrements en les oblitérant d'un caractère High-Value (FF hexadécimal) en première position de la clef de recherche, mais en fait ceci est impossible et cause une condition INVALID KEY.

REMARQUE : Comme pour READ, la clause INVALID KEY peut être omise mais par sécurité nous recommandons de la considérer obligatoire.

CLØSE Nom-de-Fichier [UNIT] [WITH LØCK]

Tout fichier en organisation directe doit être fermé en fin de travail pour permettre au système d'exploitation d'assurer les contrôles nécessaires.

Les options [UNIT] et [WITH LØCK] sont peu utilisées.

[UNIT] n'a de réelle utilité que dans le cas de fichiers multivolumes. Par exemple, lorsque pour faciliter des ajouts ultérieurs le programmeur désire répartir à peu près équitablement les 40 000 enregistrements d'un fichier sur deux volumes il spécifiera CLØSE Nom de Fichier UNIT au bout de 20 000 enregistrements, ce qui entraînera un préformatage total du premier volume et un positionnement du prochain WRITE sur la première piste du second volume.

[WITH LØCK] est une précaution annexe permettant, après fermeture du fichier, de demander au système d'exploitation de s'assurer qu'aucune réouverture intempestive du fichier n'aura lieu dans la suite de ce même programme.

Tableau résumé des options utilisées en organisation directe

Méthodes d'accès	ACCESS	Traitement	Ouverture ØPEN	Verbes	Fermeture CLØSE
B. S. A. M.	SEQUENTIAL	Lecture	INPUT	READ [INTØ]----- AT END	[UNIT] [WITH LØCK]
		Écriture (création)	ØUTPUT	WRITE [FRØM] INVALID KEY	
B. D. A. M.	RANDØM	Lecture	INPUT	SEEK READ [INTO]----- INVALID KEY	[WITH LØCK]
		Écriture (création)	ØUTPUT	SEEK WRITE [FRØM]------ INVALID KEY	
		Lecture & Écriture	I – Ø	SEEK READ [INTØ]------- INVALID KEY WRITE [FRØM]------ INVALID KEY REWRITE [FRØM]---- INVALID KEY	

Les principes généraux de programmation des fichiers en organisation directe étant donnés, nous allons voir des exemples de réalisation pratique.

CRÉATION D'UN FICHIER EN ORGANISATION DIRECTE

Un fichier en Organisation Directe sera créé séquentiellement (SEQUENTIAL) ou en accès sélectif (RANDØM) suivant la configuration de l'indicatif du fichier émetteur.

CRÉATION SEQUENTIELLE :

La création séquentielle sera toujours préférable puisqu'elle permettra, de même, plus tard des traitements séquentiels dans la séquence de l'indicatif aussi bien que des accès au hasard. Ceci suppose que le fichier séquentiel émetteur soit ordonné et surtout suppose que l'indicatif du fichier séquentiel donne directement l'ACTUAL KEY, ce qui est rarement le cas dans la pratique.

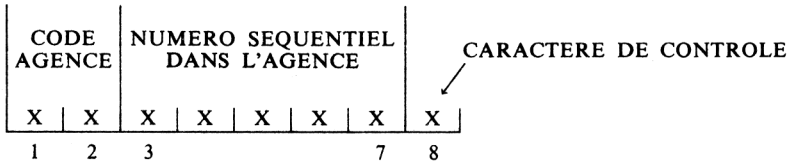
En principe, la création doit commencer avec la valeur zéro dans le numéro de piste de l'ACTUAL KEY mais le programmeur peut très bien y mettre une valeur initiale X, ce qui supposera X pistes préformatées non utilisées en début de fichier.

Dès le premier ordre d'écriture donné, le programmeur n'a même plus besoin de s'inquiéter du numéro de piste qui sera incrémenté automatiquement de 1 lorsqu'une piste sera pleine. Le programmeur a cependant toute latitude pour laisser des pistes vides pour des ajouts ultérieurs en imposant des valeurs de numéro de piste supérieures.

A noter qu'une erreur de programmation faisant régresser le numéro de piste n'aurait aucune influence car dans ce cas COBOL repositionne le numéro de piste de l'ACTUAL KEY à la juste valeur. Par contre, le programmeur devra veiller attentivement aux enregistrements doubles car rien n'empêche de créer successivement deux ou trois fois de suite le même enregistrement logique possédant la même clef de recherche.

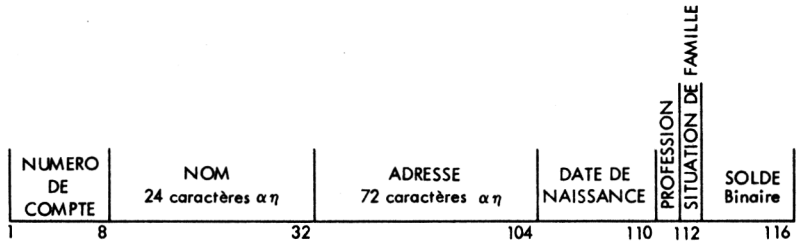
Exemple : Supposons le fichier des 400 000 numéros de comptes clients d'une société dont la croissance globale annuelle est voisine de 10 % mais variable suivant les agences.

Les numéros de comptes sont composés de la manière suivante :



Il y a une vingtaine d'agences et pour chacune, les numéros sont séquentiels à partir de 00001. Il arrive que certains numéros soient absents en attendant d'être réaffectés car les comptes clos doivent être gardés cinq ans et peuvent être réattribués après ce laps de temps.

L'enregistrement logique du fichier émetteur séquentiel bande comporte 116 octets.



La table de capacité disque 2314 nous indique que nous pourrions disposer par piste 27 enregistrements de 123 octets chacun, ce qui impose plus de 16 000 pistes donc plus de quatre volumes disques 2314.

Nous sommes au-delà des possibilités financières de cet établissement et décidons de compliquer un peu la programmation pour économiser la place sur disque. Pour celà, nous groupons les enregistrements par 10, ce qui autorise 5 enregistrements physiques de 1 288 octets ou 50 enregistrements logiques de 128 octets par piste. Le volume disque nécessaire est alors réduit de près de la moitié.

Nous prendrons une clef de 8 octets afin de respecter la règle des frontières de mots et double-mots. Un enregistrement physique comportera donc 1 288 octets répartis en une clef de 8 octets plus 10 enregistrements logiques de 128 octets chacun.

Le processus de création ou recherche sera simple : la clef de recherche étant la dizaine de numéros de comptes. Par exemple |3 3|1 2 3 4 0|0| sera la clef de l'enregistrement physique comportant les enregistrements logiques d'indicatifs

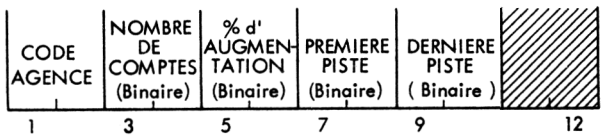
3 3	1 2 3 4 0		3 3	1 2 3 4 1		3 3	1 2 3 4 2		3 3	1 2 3 4 3
3 3	1 2 3 4 4		3 3	1 2 3 4 5		3 3	1 2 3 4 6		3 3	1 2 3 4 7
3 3	1 2 3 4 8		3 3	1 2 3 4 9

Les agences ayant une importance et une croissance variables, nous utiliserons la dernière piste pour y placer un enregistrement décrivant les caractéristiques de chacune des agences et ayant pour clef de recherche

|9 9|9 9 9 9 0 0|

Cet enregistrement comportera aussi 1 288 octets sous la forme de 99 fois une zone de 12 octets plus un FILLER de 92 octets.

Chaque zone comportant :

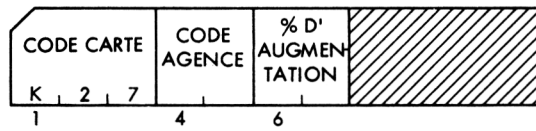


Dans le cas le plus général, le numéro de piste sera calculé par la formule :

$$\text{Numéro de Piste} = \text{Numéro de 1re Piste d'Agence} + \frac{\text{Numéro séquentiel du compte}}{50}$$

Mais dans notre cas de création séquentielle, il suffira de laisser faire COBOL qui fera progresser automatiquement le numéro de piste et nous fournira toujours ce numéro de piste dans l'ACTUAL KEY après chaque ordre WRITE.

Les pourcentages d'augmentation à prévoir sous forme d'enregistrements pré-formatés sont lus sur des cartes comme suit :



La figure 1 représente l'ordinogramme de traitement adopté pour notre création séquentielle et dont voici le programme.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. CREATION.
REMARKS. PROGRAMME DE CREATION SEQUENTIELLE  D 'UN  FICHIER
         EN ORGANISATION DIRECTE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FICART  ASSIGN TO   UT-S-FICART.
    SELECT  FIBAND  ASSIGN TO   UT-S-FIBAND.
    SELECT  FIDISQ  ASSIGN TO   DA-D-FIDISQ
                    ACCESS IS SEQUENTIAL
                    ACTUAL KEY IS CLEGROUP
                    TRACK-LIMIT  IS  900.
DATA  DIVISION.
FILE  SECTION.
FD  FICART  LABEL RECORD OMITTED
            RECORDING MODE F
            DATA RECORD CARTE.
01  CARTE.
    02  COCAR   PIC X(3).
    02  AGENCAR PIC 99.
    02  POURCAR PIC 99.
    02  FILLER  PIC X(73).
FD  FIBAND  LABEL RECORD STANDARD
            RECORDING MODE F
            BLOCK CONTAINS  3596
            DATA RECORD BANDE.
01  BANDE.
    02  NUCOMPTE.
        04  AGENBAND     PIC 99.
        04  SEQBAND      PIC 9(5).
        04  LETTRE       PIC X.
```

```
    02  SUITE              PIC X(108).
FD  FIDISQ  LABEL RECORD STANDARD
            RECORDING MODE F
            DATA RECORD DISQUE.
01  DISQUE.
    02  ENREG   OCCURS 10.
        04  INDBAND      PIC X(6).
        04  UNITBAND     PIC 9.
        04  RESTBAND     PIC X(121).
WORKING-STORAGE  SECTION.
77  IND PIC S9(5)  COMP VALUE ZERO.
77  J   PIC S9(5)  COMP VALUE ZERO.
77  K   PIC S9(5)  COMP VALUE ZERO.
01  CLEGROUP.
    02  PISTE   PIC S9(5) COMP VALUE ZERO.
    02  CLEREC.
        04  AGDISQ  PIC 99    VALUE ZERO.
        04  SEKENS  PIC 9(4)  VALUE ZERO.
        04  FILLER  PIC 99    VALUE ZERO.
01  BLOCFIN.
    02  CARACTERIS  OCCURS  99.
        04  AGENCE  PIC 99.
        04  NBCOMP  PIC S9(5) COMP.
        04  POURCEN  PIC  99.
        04  PRIMPIST PIC  S9(5) COMP.
        04  DERPIST  PIC  S9(5) COMP.
        04  FILLER   PIC  XX.
    02  FILLER       PIC  X(92) VALUE SPACE.
01  ZOTRA.
    02  COMPTRA.
        04  AGENTRA PIC 99.
        04  SEQTRA  PIC 9(5).
        04  RESEQ   REDEFINES  SEQTRA.
            06  INDIC   PIC 9(4).
            06  UNITE   PIC 9.
        04  LETRA   PIC X.
    02  SUITRA  PIC  X(108).
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT FICART FIBAND   OUTPUT FIDISQ.
    PERFORM MISABLAN VARYING IND FROM 1 BY 1 UNTIL  IND > 99.
DEBU1.
    READ FICART AT END CLOSE FICART GO TO DEBU2.
    IF COCAR NOT = 'K27' DISPLAY 'CARTE PARAMETRE ERRONEE'
        STOP RUN.
    MOVE AGENCAR TO IND
    MOVE AGENCAR TO AGENCE (IND)
    MOVE POURCAR TO POURCEN (IND)   GO TO DEBU1.
DEBU2.
    READ FIBAND AT END DISPLAY 'BANDE VIDE'  STOP RUN.
```

```
DEBU3.
    MOVE AGENBAND TO IND     MOVE 0 TO J
    IF AGENCE (IND) = ZERO  DISPLAY 'MANQUE CARTE PARAMETRE'
          AGENBAND  STOP RUN.
    MOVE AGENBAND TO AGDISQ  MOVE PISTE TO PRIMPIST (IND).
BOUCLE.
    MOVE BANDE TO ZOTRA
    READ FIBAND   END MOVE 99 TO AGENBAND.
    IF J NOT = ZERO GO TO BATIBLOC.
    MOVE INDIC TO SEKENS.
BATIBLOC.
    IF J = UNITE GO TO MOUVE.
    IF J < UNITE GO TO MANQUE.
    DISPLAY 'ENREGISTREMENT EN DOUBLE'  STOP RUN.
MANQUE.
    COMPUTE  K = J + 1
    MOVE J TO UNITBAND (K)        ADD 1 TO J
    MOVE SEKENS TO INDBAND (J)
    MOVE HIGH-VALUE  TO RESTBAND (J).
MANQU2.
    IF J = 10 GO TO ECRIT ELSE GO TO BATIBLOC.
MOUVE.
    ADD 1 TO J   MOVE ZOTRA TO ENREG (J)
    IF J NOT = 10  GO TO RUPAGEN.
ECRIT.
    WRITE DISQUE INVALID KEY DISPLAY 'CLE INVALIDE'
        CLEGROUP     STOP RUN.
    MOVE ZERO TO J.
RUPAGEN.
    IF AGENBAND = AGENTRA GO TO BOUCLE.
    IF J NOT = ZERO PERFORM MANQUE UNTIL J = 10
        PERFORM ECRIT.
    MOVE SEQTRA TO NBCOMP (IND)
    COMPUTE  DERPIST (IND) = PRIMPIST (IND)  +
          ((PISTE - PRIMPIST (IND)) * POURCEN (IND))  / 100
    PERFORM FORMAT UNTIL PISTE = DERPIST (IND) + 1
    IF AGENBAND  NOT = 99 GO TO DEBU3.
FINTRA.
    MOVE '99999900' TO CLEREC
    WRITE DISQUE FROM BLOCFIN   INVALID KEY DISPLAY
          'DERNIER BLOC INVALIDE'.
    DISPLAY  'FIN DE TRAVAIL CREATION'
    DISPLAY  'DERNIERE PISTE'  PISTE
    CLOSE FIBAND FIDISQ     STOP RUN.
MISABLAN.
    MOVE ZERO TO AGENCE (IND)  NBCOMP (IND)  POURCEN (IND)
                 PRIMPIST (IND)  DERPIST (IND).
FORMAT.
    ADD 1 TO SEKENS
    PERFORM MANQUE UNTIL J = 10
    PERFORM ECRIT.
```

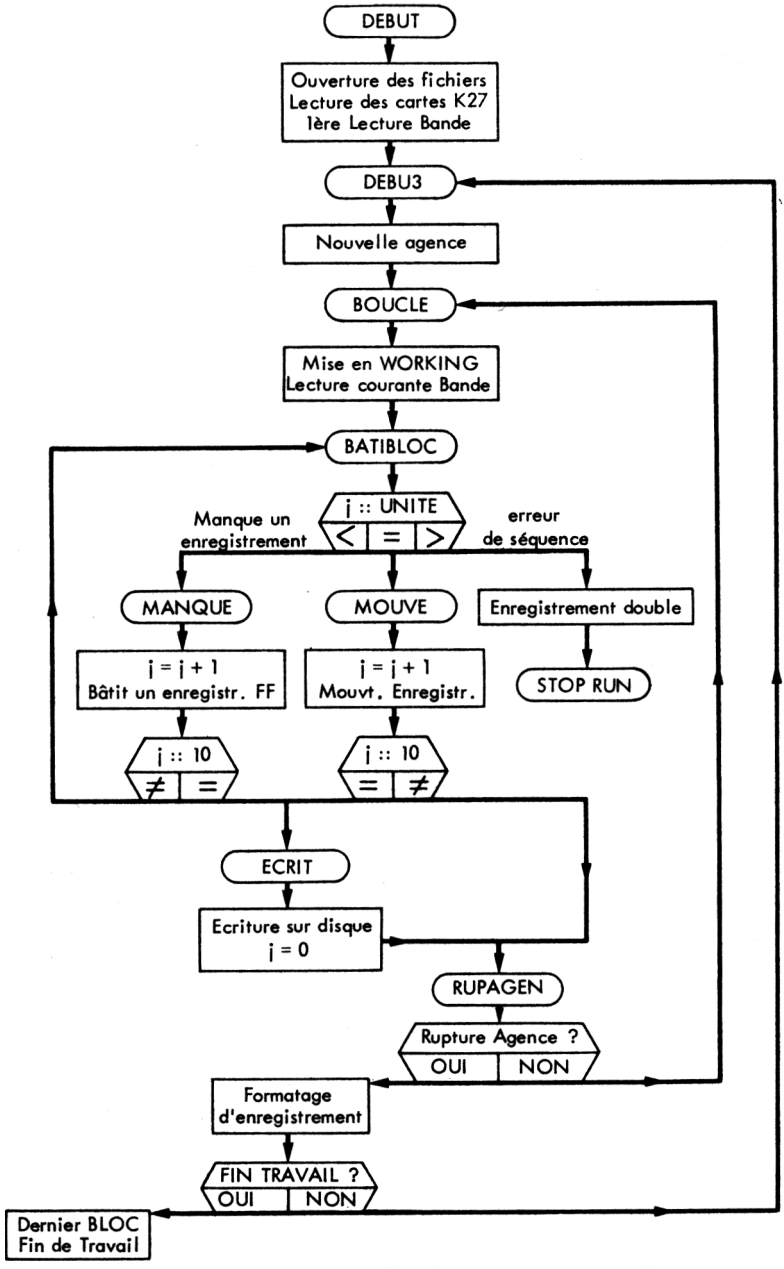

DEBUT
Ouverture des fichiers
Lecture des cartes K27
1ère Lecture Bande
DEBU3
Nouvelle agence
BOUCLE
Mise en WORKING
Lecture courante Bande
BATIBLOC
j :: UNITE
Manque un enregistrement
<
=
>
erreur de séquence
MANQUE
MOUVE
Enregistrement double
j = j + 1
Bâtit un enregistr. FF
j = j + 1
Mouvt. Enregistr.
STOP RUN
j :: 10
≠
=
j :: 10
=
≠
ECRIT
Ecriture sur disque
j = 0
RUPAGEN
Rupture Agence ?
OUI
NON
Formatage d'enregistrement
FIN TRAVAIL ?
OUI
NON
Dernier BLOC
Fin de Travail

FIG. 1.

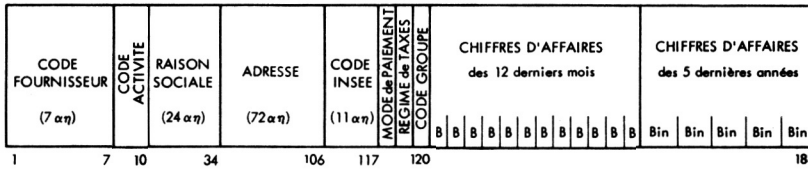
CRÉATION EN ACCÈS SÉLECTIF (RANDØM)

La création en accès sélectif sera utilisée lorsque le passage de l'indicatif d'enregistrement à l'ACTUAL KEY n'est pas immédiat, c'est-à-dire lorsque l'ordre du fichier résultant en organisation directe sera totalement différent de l'ordre du fichier séquentiel initial.

Le chargement de l'ACTUAL KEY (piste et clef de recherche) est ici absolument indispensable puisque les enregistrements seront écrits en fonction de son contenu. De même, le programmeur devra obligatoirement utiliser la clause TRACK-LIMIT car, en création RANDØM, COBOL formate les pistes allouées dès l'ordre ØPEN d'ouverture du fichier. Si un nombre insuffisant de pistes était alloué en clause TRACK-LIMIT, il serait alors impossible de procéder à des ajouts ultérieurs.

Exemple : Supposons un fichier utilisé dans une application informatique de paiement des fournisseurs. Ce fichier comporte près de 6 000 enregistrements, purgés chaque année des fournisseurs avec lesquels il n'a pas été réalisé de chiffre d'affaire depuis quatre ans.

L'enregistrement logique est composé de la manière suivante :



Le code fournisseur comporte :

— le matricule proprement dit : 6 caractères numériques.
— un caractère de contrôle alphanumérique.

La consultation des tables de capacité nous informe qu'il est possible de placer 21 enregistrements de 198 octets chacun par piste de disque IBM 2314.

La principale difficulté consiste à trouver un algorithme de calcul des numéros de piste à partir de l'indicatif car il faut éviter ce qu'on a coutume d'appeler des synonymes, c'est-à-dire éviter que plus de 21 enregistrements soient affectés à une même piste.

Afin de permettre des ajouts ultérieurs, nous prenons une marge de sécurité voisine de 20 % ce qui fait 7 000 enregistrements ou 334 pistes.

L'algorithme de calcul peut être obtenu de la manière suivante : On divise le matricule par 6997, nombre premier immédiatement inférieur à 7000, puis le reste est divisé par 21 pour obtenir un quotient égal au numéro de piste.

Par exemple, pour le matricule 59461 K

59461 : 6997 = 8 reste 3485
3485 : 21 = 165

la nominal KEY sera donc : piste 165 et clef de recherche 59461 K.

Ici aussi nous alignerons la longueur de la clef sur 8 octets d'où un FILLER de 2 octets en fin de chaque enregistrement logique

(8 + 188 + 2 = 198).

D'où le programme de création en accès sélectif :

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. CREDIREC.
REMARKS. PROGRAMME DE CREATION RANDOM D 'UN FICHIER
         EN ORGANISATION DIRECTE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FIBAND  ASSIGN TO   UT-S-FIBAND.
    SELECT  FIDISQ  ASSIGN TO   DA-D-FIDISQ
                    ACCESS IS RANDOM
                    ACTUAL KEY IS CLEF
                    TRACK-LIMIT IS 340.
DATA  DIVISION.
FILE  SECTION.
FD  FIBAND  LABEL RECORD STANDARD
            RECORDING MODE F
            BLOCK CONTAINS 3572
            DATA RECORD BANDE.
01  BANDE.
    02  MATRICULE.
        04  NUMERFOU     PIC 9(6).
        04  CARACTER     PIC X.
    02  FILLER    PIC  X(181).
FD  FIDISQ  LABEL RECORD STANDARD
            RECORDING MODE F
            DATA RECORD DISQUE.
01  DISQUE  PIC  X(190).
WORKING-STORAGE  SECTION.
77  QUOTIENT     PIC S9(5) COMP.
77  RESTE        PIC S9(5) COMP.
01  CLEF.
    02  PISTE   PIC    S9(5) COMP.
    02  CLEREC  PIC    X(7).
    02  FILLER  PIC    XX    VALUE SPACE.
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT FIBAND  OUTPUT FIDISQ.
LECTURE.
    READ FIBAND AT END CLOSE FIDISQ FIBAND
        DISPLAY  'FIN DE TRAVAIL'  STOP RUN.
    DIVIDE 6997  INTO NUMERFOU GIVING QUOTIENT REMAINDER RESTE
    DIVIDE 21  INTO RESTE  GIVING PISTE
    MOVE  MATRICULE TO CLEREC
    WRITE DISQUE FROM BANDE  INVALID KEY GO TO ERREUR.
    GO TO LECTURE.
ERREUR.
    DISPLAY  'CLEF INVALIDE POUR  '  MATRICULE
    GO TO LECTURE.
```

MISE A JOUR EN ACCÈS SÉLECTIF (UPDATE)

La procédure sera identique à la création de fichier en accès sélectif RANDØM ; à partir du fichier mouvement, le programmeur doit bâtir l'ACTUAL KEY pour rechercher l'enregistrement qu'il faut créer, modifier ou supprimer sur le fichier permanent.

La seule véritable originalité réside dans l'utilisation de la clause INVALID KEY ; en effet, pour tout enregistrement du fichier mouvement, on fera systématiquement une lecture du fichier permanent pour contrôler la validité des mouvements.

Si CRÉATION : On doit avoir INVALID KEY en lecture car l'enregistrement correspondant ne peut exister au fichier permanent.

Si MODIFICATION ou SUPPRESSIØN : On ne doit pas avoir INVALID KEY en lecture car l'enregistrement existe au fichier permanent.

On n'oubliera pas les options ØPTCD et LIMCT des cartes de contrôle J C L du système d'exploitation pour poursuivre les recherches aussi loin que nécessaire sur les pistes suivantes et surtout pour permettre l'écriture de synonymes s'il en existe.

Exemple : Nous désirons mettre à jour le fichier fournisseur de l'exemple précédent. Les enregistrements logiques du fichier Mouvement sont supposés contrôlés et de format identique à ceux du fichier disque. Longueur 190 octets avec en position 190 le code mouvement C = création, M = modification ou S = suppression.

En cas de modification, les zones à modifier sont caractérisées par le fait qu'elles ne sont pas blanches sur l'enregistrement mouvement.

En suppression, on reécrit l'enregistrement avec un S en position 190.

On remarquera l'utilisation de WRITE (création) et REWRITE (modification, suppression) en relation avec W en clause SELECT.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. DIRECMAJ.
REMARKS. PROGRAMME DE MISE A JOUR ACCES SELECTIF (RANDOM) D 'UN
         FICHIER EN ORGANISATION DIRECTE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  BANDMOUV     ASSIGN TO   UT-S-BANDMOUV.
    SELECT  DISQPERM     ASSIGN TO   DA-W-DISQPERM
                         ACCESS IS RANDOM
                         ACTUAL KEY IS CLEF.
DATA  DIVISION.
FILE  SECTION.
FD  BANDMOUV     LABEL RECORD STANDARD
                 RECORDING MODE F
                 BLOCK CONTAINS 3610
                 DATA RECORD BANDE.
01  BANDE.
    02  BMATR.
        04  BNUMFOU PIC  9(6).
        04  BCARAC  PIC  X.
    02  BACTIV      PIC  XXX.
    02  BRAISOC     PIC  X(24).
    02  BADDR       PIC  X(72).
    02  BINSEE      PIC  X(11).
    02  BPAIE       PIC  X.
    02  BTAXE       PIC  X.
    02  BGROUPE PIC X.
    02  BCHAFM  OCCURS  12  PIC S9(9) COMP.
    02  BCHAFAN OCCURS   5  PIC S9(9) COMP.
    02  FILLER  PIC X.
    02  BCODE   PIC X.
FD  DISQPERM     LABEL RECORD STANDARD
                 RECORDING MODE F
                 DATA RECORD DISQUE.
01  DISQUE.
    02  MATR.
        04  NUMFOU  PIC  9(6).
        04  CARAC   PIC  X.
    02  ACTIV   PIC    XXX.
    02  RAISOC  PIC    X(24).
    02  ADDR    PIC    X(72).
    02  INSEE   PIC    X(11).
    02  PAIE    PIC    X.
    02  TAXE    PIC    X.
    02  GROUPE  PIC    X.
    02  CHAFM   OCCURS  12  PIC S9(9) COMP.
    02  CHAFAN  OCCURS   5  PIC S9(9) COMP.
    02  FILLER  PIC XX.
WORKING-STORAGE  SECTION.
77  RESTE       PIC S9(5) COMP.
77  QUOTIENT    PIC S9(5) COMP.
77  I           PIC S9(5) COMP.
01  CLEF.
    02  PISTE   PIC  S9(5) COMP.
    02  CLEREC  PIC  X(7).
    02  FILLER  PIC  XX VALUE SPACE.
```

```
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT BANDMOUV  I-O DISQPERM.
LECTURE.
    READ BANDMOUV AT END CLOSE BANDMOUV DISQPERM
        DISPLAY 'FIN DE TRAVAIL'  STOP RUN.
CALCLE.
    DIVIDE 6997 INTO BNUMFOU GIVING QUOTIENT REMAINDER RESTE
    DIVIDE 21 INTO  RESTE GIVING PISTE
    MOVE BMATR TO CLEREC.
LECDIS.
    READ DISQPERM INVALID KEY GO TO CREATION.
    IF BCODE = 'C' GO TO ERREUR1.
    IF BCODE = 'S' GO TO SUPPRESSION.
MODIFICATION.
    IF BACTIV   NOT = SPACE MOVE    BACTIV  TO  ACTIV.
    IF BRAISOC  NOT = SPACE MOVE    BRAISOC TO  RAISOC.
    IF BADDR    NOT = SPACE MOVE    BADDR   TO  ADDR.
    IF BINSEE   NOT = SPACE MOVE    BINSEE  TO  INSEE.
    IF BPAIE    NOT = SPACE MOVE    BPAIE   TO  PAIE.
    IF BTAXE    NOT = SPACE MOVE    BTAXE   TO  TAXE.
    IF BGROUPE  NOT = SPACE MOVE    BGROUPE TO  GROUPE.
    PERFORM MOIS VARYING I FROM 1 BY 1 UNTIL I > 12
    PERFORM AN   VARYING I FROM 1 BY 1 UNTIL I > 5
    REWRITE DISQUE INVALID KEY GO TO ERREUR2.
    GO TO LECTURE.
SUPPRESSION.
    REWRITE DISQUE FROM BANDE INVALID KEY GO TO ERREUR2.
    GO TO LECTURE.
CREATION.
    IF BCODE NOT = 'C' GO TO ERREUR1.
    MOVE SPACE  TO  BCODE.
    WRITE DISQUE FROM BANDE  INVALID KEY GO TO ERREUR2.
    GO TO LECTURE.
ERREUR1.
    DISPLAY 'ERREUR DE MISE A JOUR'
    DISPLAY  BMATR  BCODE
    GO TO LECTURE.
ERREUR2.
    DISPLAY 'ECRITURE IMPOSSIBLE'
    DISPLAY  BMATR  BCODE
    GO TO LECTURE.
MOIS.
    IF BCHAFM (I) NOT = ZERO  MOVE BCHAFM (I) TO CHAFM (I).
AN.
    IF BCHAFAN (I) NOT = ZERO MOVE BCHAFAN (I) TO CHAFAN (I).
```

CHAPITRE II

L'ORGANISATION RELATIVE (1)

PRINCIPES GÉNÉRAUX

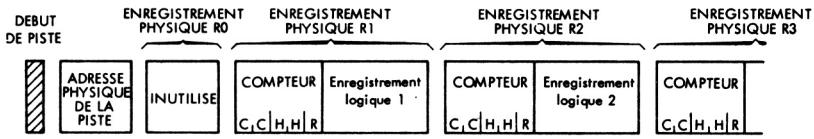
L'organisation relative consiste à désigner un enregistrement logique de fichier sur mémoire à accès sélectif par le numéro relatif d'enregistrement, compté à partir de zéro pour le premier enregistrement du fichier. Un enregistrement logique est donc *défini exclusivement par son numéro d'ordre.*

L'organisation relative est la plus performante des trois organisations mais sera difficile à réaliser dans la pratique car il est assez rare de pouvoir désigner un enregistrement par son numéro d'ordre.

Dans cette organisation, seuls les enregistrements de *longueur fixe non bloqués* sont autorisés car COBOL traduit le numéro relatif d'enregistrement en adresse disque.

C C H H R avec C C = numéro de cylindre
H H = numéro de tête de lecture (ou de piste dans le cylindre)
R = numéro d'enregistrement dans la piste.

Une piste physique est organisée de la manière suivante :



Enregistrement physique Ro : n'est pas utilisé car seul le format fixe est autorisé.

(1) L'ensemble de ce chapitre est hors normes A. N. S. et valable seulement en système d'exploitation OS.

Enregistrement physique R1, R2, ... : chacun des enregistrements est composé de deux parties :

— Le compteur adresse disque :

C C = 2 octets en binaire
H H = 2 octets en binaire
R = 1 octet en binaire (ce qui suppose un maximum de 255 enregistrements par piste).

— L'enregistrement logique proprement dit.

L'argument de positionnement ou recherche d'un enregistrement logique sera fourni à COBOL dans une zone de donnée NØMINAL KEY définie en WØRKING-STØRAGE SECTIØN sous forme d'une zone binaire sur 4 octets de PICTURE S9(8), maximum 16 777 215.

Par exemple :

```
A       B
77      CLEF PIC S9(8) CØMP.
```

FORMAT GÉNÉRAL D'UN PROGRAMME

Le format général d'un programme en Organisation Relative est le suivant :

```
A       B
 IDENTIFICATIØN DIVISIØN.
 ENVIRØNMENT DIVISIØN.
 CØNFIGURATIØN SECTIØN.
 INPUT-ØUTPUT SECTIØN.
  FILE-CØNTRØL.
                SELECT Nom-de-Fichier
                              ASSIGN TØ Nom-de-Fichier-Système
                ACCESS   MØDE IS { SEQUENTIAL }
                                 { RANDØM     }
                NØMINAL KEY   IS  Nom-Donnée-1.
 DATA DIVISIØN.
  FILE SECTIØN.
FD      Nom-de-Fichier
                                  } Description du Fichier
```

```
A       B
WØRKING-STØRAGE SECTIØN.
77      Nom-Donnée-1 PIC S9(8) CØMP.
PRØCEDURE DIVISIØN.
        ØPEN { INPUT  }
             { ØUTPUT } Nom-de-Fichier
             { I-Ø    }
        [ READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
                                     { AT END     }
                                     { INVALID KEY } Ordre Impératif ]
        [ { WRITE   }
          { REWRITE } Nom d'Enregistrement [FRØM Nom-Donnée]
                      INVALID KEY Ordre Impératif ]
          CLØSE Nom-de-Fichier [UNIT] [WITH LØCK].
```

LES INSTRUCTIONS COBOL

SELECT Nom-de-Fichier de Data Division
ASSIGN TØ Nom-de-Fichier-Système

Le nom de fichier Système sera écrit sous la forme :

DA-R- | Nom-de-Fichier Symbolique de JCL |

Le R désignant l'organisation relative.

Exemple :

SELECT FICØMPTE ASSIGN TØ DA-R-FICØMPTE

ACCESS MØDE IS { SEQUENTIAL }
               { RANDØM   }

Les deux méthodes d'accès utilisées sont encore :

BSAM pour SEQUENTIAL
BDAM pour RANDØM

SEQUENTIAL s'adressera à la création d'un fichier Organisation Relative à partir d'un fichier séquentiel pur ou pour la lecture d'un fichier en Organisation Relative.

RANDØM ne sera utilisé que pour la lecture ou l'écriture en accès aléatoire car la création RANDØM d'un fichier en Organisation Relative est interdite.

NØMINAL KEY IS Nom-Donnée-1

Cette clause permettra de définir à COBOL dans quelle zone de donnée se trouve le numéro relatif d'enregistrement à lire ou écrire.

Nous verrons toutefois en cas de création de fichier que cette clause peut éventuellement être omise (mais non sans risques...).

ØPEN { INPUT / ØUTPUT / I-Ø } Nom-de-Fichier

Les fichiers en organisation relative devront être ouverts en :

INPUT avant le premier READ de lecture
ØUTPUT avant le premier ordre WRITE d'écriture
I-Ø s'il s'agit d'un fichier en lecture-écriture utilisant les verbes READ et REWRITE.

READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
{ AT END / INVALID KEY } Ordre Impératif

Lorsqu'un fichier en Organisation Relative est lu séquentiellement, on utilise simplement la clause AT END Ordre Impératif.

Par contre, en traitement aléatoire (RANDØM) il faut utiliser l'option INVALID KEY Ordre Impératif qui provoque un débranchement lorsque le numéro relatif d'enregistrement fourni dans la NØMINAL KEY est hors des limites du fichier.

Notons au passage que les enregistrements préformatés inutilisés (voir CLØSE et création) pourront être lus.

{ WRITE / REWRITE } Nom-d'Enregistrement [FRØM Nom-de-zone]
INVALID KEY Ordre Impératif.

En organisation relative nous aurons une réelle différenciation des effets des ordres WRITE et REWRITE.

WRITE s'adressera à la création séquentielle de fichiers en organisation relative tandis que REWRITE sera réservé aux fichiers ouverts en I-Ø donc en lecture et écriture.

La clause INVALID KEY peut être omise mais est indispensable dans la réalité pour prévenir un chargement intempestif dans la NOMINAL KEY d'un numéro d'enregistrement sortant des limites du fichier.

CLØSE Nom-de-Fichier [UNIT] [WITH LØCK]

Tout fichier en organisation relative doit être fermé en fin de travail pour permettre au système d'Exploitation d'assurer les contrôles nécessaires. Ici aussi les clauses [UNIT] et [WITH LØCK] sont peu utilisées :

[UNIT] peut être utilisé dans le cas de fichiers multivolumes et permet en donnant l'ordre CLØSE... UNIT de clore un volume de fichier avant qu'il ne soit complet pour faire réaliser automatiquement par COBOL un préformatage des enregistrements de fin de volume destinés à des ajouts ultérieurs. En outre, le prochain ordre WRITE (ou READ) est positionné sur la première piste du volume suivant.

[WITH LØCK] est une précaution complémentaire permettant de demander au système d'exploitation de s'assurer, après fermeture du fichier, qu'aucune réouverture intempestive de ce fichier n'aura lieu dans la suite de ce même programme.

Signalons qu'en création, l'ordre CLØSE déclenche un formatage des enregistrements de fin de la dernière piste écrite (et de tous les enregistrements sautés si le contenu de la NØMINAL KEY a été accru depuis le dernier ordre WRITE). Un enregistrement inutilisé formaté se distingue des autres par High-Value (FF hexadécimal) dans le premier octet de l'enregistrement.

Tableau résumé des options utilisées en organisation relative

Méthodes d'accès	Access	Traitement	Ouverture ØPEN	Verbes	Fermeture CLØSE
B. S. A. M.	SEQUENTIAL	Lecture	INPUT	READ [INTØ] AT END	[UNIT]
		Écriture (création)	ØUTPUT	WRITE [FRØM] INVALID KEY	[WITH LØCK]
B. D. A. M.	RANDØM	Lecture	INPUT	READ [INTØ] INVALID KEY	[WITH LØCK]
		Lecture & écriture	I-Ø	READ [INTØ] INVALID KEY REWRITE [FRØM] INVALID KEY	

Après l'énoncé des principes généraux de traitement qui précède, nous allons voir des exemples de création et mise à jour de fichiers en organisation relative.

CRÉATION D'UN FICHIER EN ORGANISATION RELATIVE

Rappelons que seule la création séquentielle (SEQUENTIAL) sera possible pour un fichier en organisation relative.

La création commence théoriquement avec la valeur zéro dans la NØMINAL KEY c'est-à-dire par l'enregistrement numéro zéro du fichier, mais le programmeur peut s'il le désire commencer avec le numéro X, ce qui entraînera la création par COBOL de X enregistrements préformatés non utilisés en début de fichier.

Par mesure de précaution, nous recommandons de ne pas utiliser le premier octet des enregistrements logiques afin de le réserver au caractère High-Value indiquant qu'un enregistrement est formaté mais non utilisé.

Dès le premier ordre d'écriture donné, il n'y a plus à s'inquiéter du numéro relatif de piste qui sera automatiquement incrémenté de un après chaque ordre WRITE. Mais il sera toujours possible de laisser de la place disponible sous forme d'enregistrements préformatés pour des ajouts ultérieurs en imposant des valeurs de numéros d'enregistrements supérieures.

Le programmeur devra d'ailleurs veiller tout particulièrement en fin de fichier à faire formater, par l'ordre CLØSE, un nombre d'enregistrements suffisant pour des ajouts ultérieurs en imposant une valeur de NØMINAL KEY assez élevée.

Comme en organisation directe, une erreur de programmation faisant régresser le numéro relatif d'enregistrement est automatiquement corrigée par COBOL qui repositionne la valeur de la NØMINAL KEY à sa juste valeur.

Exemple : Nous désirons créer le fichier stock d'une Société de Vente par correspondance. Les numéros d'article sont numériques avec 6 caractères et, compte tenu des tailles-couleurs, etc., il peut exister 999 999 articles au catalogue. Nous désirons que ce fichier n'occupe si possible qu'une seule unité 2314 (8 volumes) d'où 32 enregistrements de 125 octets par piste d'après la table de capacité (sans clef).

La création de fichier sera très simple puisque COBOL assure automatiquement le formatage des enregistrements manquants ; il suffira de mouvementer le code article dans la NOMINAL KEY avant chaque écriture. Nous contrôlerons cependant en fin de fichier que l'article numéro 999999 a bien été écrit.

HIGH-VALUE	NUMERO ARTICLE	DESIGNATION	TAILLE	COLORI	PRIX DE VENTE	QTE STOCK	STOCK MINI	PRIX ACHAT	DELAI LIVRAISON	CODE FOURNISSEUR	CODE ACHETEUR	LIEU STOCKAGE	IMPUTATION	QUANTITES VENDUES DANS 12 DERNIERS MOIS	FILLER
	9 (6)	X (30)	99	X	S9(7)V99 Bin	S9(9) Bin	S9(9) Bin	S9(7)v99 Bin	9 (3) P	9 (6)	99	99	9999	12 fois S9 (9) Binaire	X (5)
1	7	37	40		44	48	52	56	58	64	66	68	72	120	125

D'où le programme de création en organisation relative :

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. CRELATIF.
REMARKS. PROGRAMME DE CREATION D'UN FICHIER
         EN ORGANISATION RELATIVE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  BANDENTR     ASSIGN TO   UT-S-BANDENTR.
    SELECT  DISQSOR      ASSIGN TO   DA-R-DISQSOR
                         ACCESS IS SEQUENTIAL
                         NOMINAL KEY IS CLEF.
DATA  DIVISION.
FILE  SECTION.
FD  BANDENTR     LABEL RECORD STANDARD
                 RECORDING MODE  F
                 BLOCK CONTAINS 3625
                 DATA RECORD ENTREE.
01  ENTREE.
    02  FILLER  PIC X.
    02  ARTICLE PIC 9(6).
    02  FILLER  PIC X(118).
FD  DISQSOR      LABEL RECORD STANDARD
                 RECORDING MODE  F
                 DATA RECORD SORTIE.
01  SORTIE  PIC X(125).
WORKING-STORAGE  SECTION.
77  CLEF     PIC S9(8)  COMP.
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT BANDENTR     OUTPUT DISQSOR.
LECTURE.
    READ BANDENTR  AT END GO TO FINTRAVAIL.
    MOVE ARTICLE TO CLEF
    WRITE SORTIE FROM ENTREE  INVALID KEY DISPLAY
          'ECRITURE  IMPOSSIBLE'   ARTICLE.
    GO TO LECTURE.
FINTRAVAIL.
    MOVE  999999 TO CLEF
    CLOSE  BANDENTR  DISQSOR
    STOP RUN.
```

MISE A JOUR (UPDATE) EN ACCÈS SÉLECTIF (RANDØM) D'UN FICHIER ORGANISATION RELATIVE

La procédure de traitement de mise à jour est semblable à celle de l'organisation directe avec toutefois une différence dans le fait que les enregistrements formatés inutilisés, avec HIGH-VALUE en première position sont accessibles. On contrôlera donc par un ordre READ de lecture que l'enregistrement existe au fichier permanent mais aussi que en cas de mouvement :

CREATION : l'enregistrement du fichier permanent comporte HIGH-VALUE en première position.

MODIFICATION ou SUPPRESSION : l'enregistrement du fichier permanent n'est pas altéré par HIGH-VALUE.

Par suite, pour supprimer un enregistrement il suffira de mettre HIGH-VALUE en première position de l'enregistrement.

Exemple : Nous désirons établir un programme de rectification du fichier stock de l'exemple précédent.

Les enregistrements du fichier mouvement sont supposés contrôlés et de format identique à ceux du fichier disque, longueur 125 octets avec le code mouvement C = Création, M = Mouvement et S = Suppression en première position de l'enregistrement.

En cas de modification, les zones non blanches de l'enregistrement mouvement prennent la place de celles du fichier permanent.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. RELATMAJ.
REMARKS. PROGRAMME DE MISE A JOUR ACCES SELECTIF (RANDOM)
         D 'UN FICHIER EN ORGANISATION RELATIVE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  BANDE   ASSIGN TO   UT-S-BANDE.
    SELECT  DISQUE  ASSIGN TO   DA-R-DISQUE
                    ACCESS IS RANDOM
                    NOMINAL KEY IS CLEF.
DATA  DIVISION.
FILE  SECTION.
FD  BANDE   LABEL RECORD STANDARD
            RECORDING MODE F
            BLOCK CONTAINS 3625
            DATA RECORD MOUVEM.
01  MOUVEM.
    02  MOCODE       PIC     X.
    02  MOARTIC      PIC     9(6).
    02  MODESIGN     PIC     X(30).
    02  MOTAILLE     PIC     XX.
    02  MOCOLOR      PIC     X.
    02  MOPV         PIC     X(4).
    02  MOQTSTOCK    PIC     X(4).
    02  MOSTMINI     PIC     X(4).
    02  MOPA         PIC     X(4).
    02  MODELIV      PIC     XX.
    02  MOFOUR       PIC     X(6).
    02  MOACH        PIC     XX.
    02  MOLIEU       PIC     XX.
    02  MOIMP        PIC     X(4).
    02  MOQTV   OCCURS 12  PIC  X(4).
    02  FILLER       PIC     X(5).
FD  DISQUE  LABEL RECORD STANDARD
            RECORDING MODE F
            DATA RECORD PERMAN.
01  PERMAN.
    02  FF       PIC     X.
    02  ARTIC    PIC     9(6).
    02  DESIGN   PIC     X(30).
    02  TAILLE   PIC     XX.
    02  COLOR    PIC     X.
    02  PV       PIC     X(4).
    02  QTSTOCK  PIC     X(4).
    02  STMINI   PIC     X(4).
    02  PA       PIC     X(4).
    02  DELIV    PIC     XX.
    02  FOUR     PIC     X(6).
    02  ACH      PIC     XX.
    02  LIEU     PIC     XX.
    02  IMP      PIC     X(4).
    02  QTV  OCCURS  12   PIC X(4).
    02  FILLER  PIC      X(5).
WORKING-STORAGE  SECTION.
77  CLEF     PIC S9(8) COMP.
77  I        PIC S9(9) COMP.
```

```
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT BANDE   I-O DISQUE.
LECTURE.
    READ BANDE  AT END CLOSE BANDE DISQUE
          DISPLAY 'FIN DE TRAVAIL'     STOP RUN.
    MOVE MOARTIC TO CLEF
    READ DISQUE INVALID KEY DISPLAY 'IMPOSSIBLE LIRE'
               MOCODE MOARTIC  GO TO LECTURE.
    IF  MOCODE = 'C'  GO TO  CREATION.
    IF  MOCODE = 'M'  GO TO  MODIFICATION.
    IF  MOCODE = 'S'  GO TO  SUPPRESSION.
    DISPLAY  'ERREUR CODE MAJ'  MOCODE MOARTIC  GO TO LECTURE.
CREATION.
    IF FF NOT = HIGH-VALUE  DISPLAY
          'ERREUR CREATION'     MOCODE MOARTIC  GO TO LECTURE.
    MOVE MOUVEM TO PERMAN   MOVE SPACE TO FF
    REWRITE PERMAN  INVALID KEY DISPLAY
          'CREATION IMPOSSIBLE'     MOCODE  MOARTIC.
    GO TO LECTURE.
MODIFICATION.
    IF FF = HIGH-VALUE  DISPLAY
          'ERREUR MODIFICATION' MOCODE MOARTIC GO TO LECTURE.
    IF  MODESIGN     NOT = SPACE MOVE    MODESIGN  TO  DESIGN.
    IF  MOTAILLE     NOT = SPACE MOVE    MOTAILLE  TO  TAILLE.
    IF  MOCOLOR      NOT = SPACE MOVE    MOCOLOR   TO  COLOR.
    IF  MOPV         NOT = SPACE MOVE    MOPV      TO  PV.
    IF  MOQTSTOCK    NOT = SPACE MOVE    MOQTSTOCK TO  QTSTOCK.
    IF  MOSTMINI     NOT = SPACE MOVE    MOSTMINI  TO  STMINI.
    IF  MOPA         NOT = SPACE MOVE    MOPA      TO  PA.
    IF  MODELIV      NOT = SPACE MOVE    MODELIV   TO  DELIV.
    IF  MOFOUR       NOT = SPACE MOVE    MOFOUR    TO  FOUR.
    IF  MOACH        NOT = SPACE MOVE    MOACH     TO  ACH.
    IF  MOLIEU       NOT = SPACE MOVE    MOLIEU    TO  LIEU.
    IF  MOIMP        NOT = SPACE MOVE    MOIMP     TO  IMP.
    PERFORM BOUCLE VARYING I FROM 1 BY 1 UNTIL I > 12
    REWRITE PERMAN  INVALID KEY DISPLAY
          'MODIFICATION IMPOSSIBLE'   MOCODE  MOARTIC.
    GO TO LECTURE.
SUPPRESSION.
    IF FF = HIGH-VALUE  DISPLAY
          'ERREUR SUPPRESSION' MOCODE MOARTIC  GO TO LECTURE.
    MOVE HIGH-VALUE TO FF
    REWRITE PERMAN INVALID KEY DISPLAY
          'SUPPRESSION IMPOSSIBLE'  MOCODE MOARTIC.
    GO TO LECTURE.
BOUCLE.
    IF MOQTV (I) NOT = SPACE    MOVE MOQTV (I) TO QTV (I).
```

CHAPITRE III

L'ORGANISATION SÉQUENTIEL INDEXÉ (1)

PRINCIPES GÉNÉRAUX

L'organisation séquentiel indexé, plus complexe en principe que les organisations que nous venons de voir, est cependant la plus utilisée car finalement la plus facile à adapter dans la pratique.

Cette organisation consiste à permettre la recherche d'un enregistrement logique par consultation de tables successives. On peut comparer ce processus de recherche à la consultation du BOTTIN. Un index général nous indique le volume à consulter puis une table des matières nous indique le chapitre, puis éventuellement la page, enfin une recherche séquentielle dans la page nous fournit le renseignement désiré.

Les consultations sont réalisées à l'aide d'une clef de recherche (NØMINAL KEY) comparée à une clef d'enregistrement (RECØRD KEY) présente physiquement dans l'enregistrement logique, et de tables cylindres comportant la valeur maximum de la clef sur chaque cylindre, enfin de tables pistes comportant la valeur maximum de la clef sur chaque piste.

Pour un fichier occupant un volume disque, on cherchera donc sur quel cylindre se trouve l'enregistrement par comparaison de la NØMINAL KEY aux valeurs maximales puis on cherchera de même à quelle piste du cylindre correspond cette clef. Enfin, on recherchera séquentiellement une égalité entre NØMINAL KEY et RECØRD KEY sur la piste.

Exemple : Si nous recherchons un enregistrement à partir d'une NØMINAL KEY = 1247 sur un fichier occupant trois cylindres.

(1) L'ensemble de ce chapitre est hors normes A. N. S.

Table des cylindres

CYL. 0	CYL. 1	CYL. 2	CYL. 3				
0805	1507	2078					

Table des pistes du cylindre 0

Piste 0	Piste 1	Piste 2	Piste 3	Piste 4	Piste 5	Piste 6	Piste 7	Piste 8	Piste 9
0032	0073	0116	0159	0198	0235	0270	0351	0363	0387
Piste 10	Piste 11	Piste 12	Piste 13	Piste 14	Piste 15	Piste 16	Piste 17	Piste 18	Piste 19
0413	0487	0503	0539	0566	0578	0639	0687	0713	0805

Table des pistes du cylindre 1

Piste 0	Piste 1	Piste 2	Piste 3	Piste 4	Piste 5	Piste 6	Piste 7	Piste 8	Piste 9
0847	0867	0899	0916	0962	0993	1017	1055	1093	1124
Piste 10	Piste 11	Piste 12	Piste 13	Piste 14	Piste 15	Piste 16	Piste 17	Piste 18	Piste 19
1169	1188	1245	1261	1295	1353	1378	1411	1462	1507

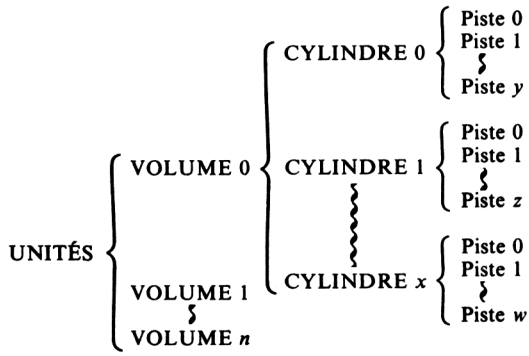
Table des pistes du cylindre 2

Piste 0	Piste 1	Piste 2	Piste 3	Piste 4	Piste 5	Piste 6	Piste 7	Piste 8	Piste 9
1525	1549	1578	1607	1649	1673	1705	1736	1767	1798
Piste 10	Piste 11	Piste 12	Piste 13	Piste 14	Piste 15	Piste 16	Piste 17	Piste 18	Piste 19
1821	1850	1889	1899	1927	1938	1972	2015	2043	2078

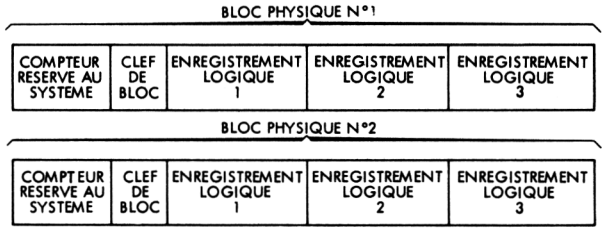
La consultation de la table des cylindres nous informe que l'enregistrement doit être sur le cylindre 1 puisque la clef maximum sur ce cylindre est $1507 > 1247$.

Puis la table des pistes du cylindre 1 nous informe que l'enregistrement est sur la piste 13 car $1245 < 1247 < 1261$.

Si le fichier occupe plusieurs volumes, on aura en plus une table des volumes ; et si le fichier occupe plusieurs unités on aura encore en plus une table unités suivant la hiérarchie :



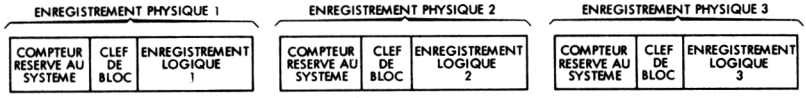
Seuls les enregistrements de format Fixe F bloqués ou non bloqués sont autorisés. Les blocs physiques ont la forme ci-dessous en format bloqué :



La clef de bloc est la duplication de la clef de l'enregistrement de niveau le plus élevé dans le bloc, ce qui est conforme au principe général du séquentiel indexé qui consiste à comparer à une valeur maximale pour savoir si la clef est comprise ou non dans un ensemble.

Chaque enregistrement logique du bloc physique contient, en plus, sa propre RECØRD KEY qui doit être située à une adresse fixe dans l'enregistrement logique. Toutefois, la RECØRD KEY ne pourra être positionnée en première position de l'enregistrement car cette position est utilisée pour y mettre HIGH-VALUE (FF hexadécimal) dans le cas d'enregistrements formatés inutilisés ou d'enregistrements éliminés.

Dans le format fixe non-bloqué on retrouvera la clef de bloc, cette information est redondante mais est due au fait que le format non bloqué est traité comme un cas particulier du format bloqué.



Il en résulte que le programmeur ne devra pas oublier la dimension de la clef de bloc dans le calcul du nombre d'enregistrements physiques qu'il est possible de disposer par piste.

Un fichier Séquentiel Indexé ne peut être créé que séquentiellement (SEQUENTIAL) à partir d'un fichier ordonné suivant l'indicatif qui tient lieu de RECØRD KEY. Pour des ajouts ultérieurs, le programmeur peut intercaler des enregistrements formatés inutilisés avec HIGH-VALUE en première position. La création de tels enregistrements n'est pas absolument obligatoire cependant car s'il ne subsiste pas assez de place pour intercaler de nouveaux enregistrements, il existe des *pistes de dépassement de capacité* (ØVERFLØW) destinées à les recevoir. Plus exactement ces pistes sont destinées à recevoir les enregistrements éjectés des pistes primaires par l'intercalement d'un nouveau.

Exemple : Si nous avons sur la piste 12 les enregistrements de clefs 1112-1113-1115-1116-1118 ; l'enregistrement de clef 1114 viendra s'intercaler dans la séquence et causera l'éjection de l'enregistrement de clef 1118 vers la piste ØVERFLØW. D'où nous aurons 1112-1113-1114-1115-1116 sur la piste 12 et 1118 sur la piste ØVERFLØW.

Par contre, si l'enregistrement 1118 était altéré par HIGH-VALUE celui-ci n'irait pas en piste ØVERFLØW mais serait simplement perdu.

REMARQUE : En séquentiel indexé, la clef d'un enregistrement doit être unique même pour les enregistrements altérés par HIGH-VALUE qui ne peuvent être créés avec des clefs identiques.

En organisation séquentiel indexé il convient donc de réserver, grâce aux cartes de contrôle JCL du système d'Exploitation, de la place disque pour trois zones :

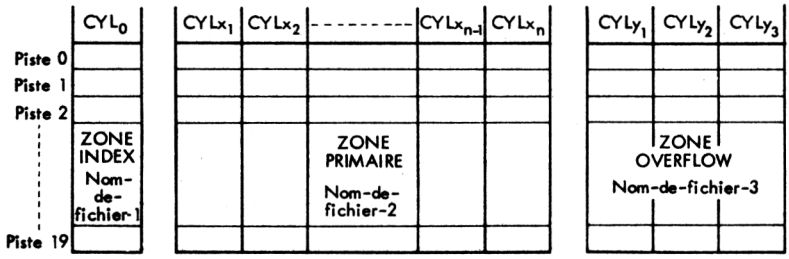
ZONE PRIMAIRE : Contenant le fichier proprement dit
ZONE OVERFLOW : Contenant les pistes de dépassement de capacité
ZONE INDEX : Pour les tables cylindres, pistes, etc.

Si le programmeur fait suivre le nom de fichier Symbolique de J C L par trois cartes DD, on aura réservation de trois zones indépendantes pour les parties primaires, overflow et index.

. Par exemple, nous réserverons trois sous-fichiers appelés Nom-de-fichier-1, Nom-de-fichier-2, Nom-de-fichier-3 pour le fichier séquentiel indexé Nom-de-fichier en écrivant :

```
// Nom-de-Fichier  DD  DSNAME = Nom-de-Fichier-1 (INDEX) ------
//                 DD  DSNAME = Nom-de-Fichier-2 (PRIME) ------
//                 DD  DSNAME = Nom-de-Fichier-3 (ØVFLØW) ----
```

Ce qui peut se schématiser de la manière suivante :

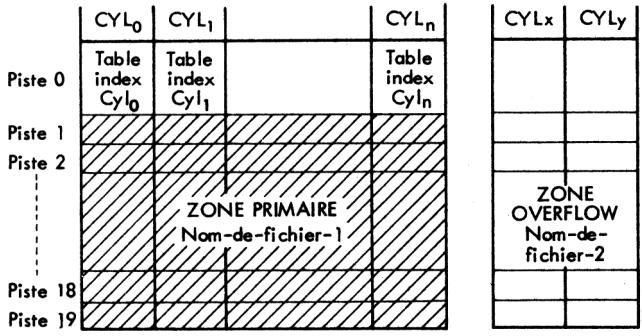


Cette solution sera intéressante à condition de placer les trois zones sur des volumes différents de manière à ne pas provoquer continuellement des déplacements de têtes de lecture, mais suppose évidemment qu'on soit suffisamment riche en volumes disques.

. Nous pourrons aussi ne réserver que deux sous-fichiers : Nom-de-fichier-1, Nom-de-fichier-2 en écrivant :

```
// Nom-de-Fichier  DD  DSNAME = Nom-de-Fichier-1 (PRIME) ------
//                 DD  DSNAME = Nom-de-Fichier-2 (ØVFLØW) ----
```

dont le résultat peut se schématiser comme suit :

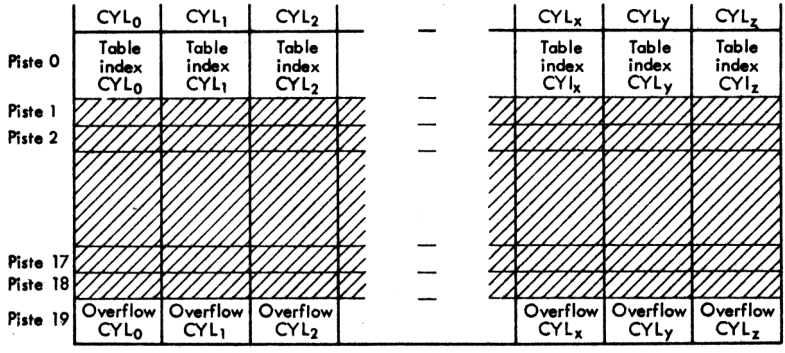


Chaque cylindre possède sa propre table index pistes, solution qui permet d'éviter des déplacements de têtes de lecture entre la recherche sur table de la piste et la recherche de l'enregistrement sur la piste.

. Enfin le programmeur pourra ne réserver qu'une seule zone de fichier avec une seule carte DD.

```
// Nom-de-Fichier  DD  DSNAME = Nom-de-Fichier (PRIME) -------
```

on ne réserve pas de zone overflow particulière mais on obtient une zone overflow par cylindre suivant le schéma ci-dessous.



Cette dernière méthode présente le désavantage de multiplier les pistes d'overflow mais par contre améliore les temps d'accès car il ne faut pas de déplacement de têtes de lecture pour aller créer ou chercher une information en zone overflow.

FORMAT GÉNÉRAL D'UN PROGRAMME

Le format général d'un programme en organisation séquentiel indexé est le suivant :

```
A      B
 IDENTIFICATIØN DIVISIØN.
                ⌇
 ENVIRØNMENT DIVISIØN.

 CØNFIGURATIØN SECTIØN.
                ⌇
 INPUT-ØUTPUT SECTIØN.

  FILE-CØNTRØL.
                SELECT Nom-de-Fichier
                ASSIGN TØ Nom-de-Fichier-Système
                [RESERVE { NØ              } ALTERNATE AREA]
                         { Nombre Entier   }

                 ACCESS    MØDE IS { SEQUENTIAL }
                                   { RANDØM     }

                 RECØRD   KEY IS    Nom-Donnée-1
                [NØMINAL  KEY IS    Nom-Donnée-2]
                [TRACK-AREA   IS { Nom-Donnée-3 }
                                 { Nombre Entier }
                                                CHARACTERS].
  I-Ø-CØNTRØL.
                [APPLY CØRE-INDEX ØN Nom-de-Fichier]
                [APPLY REØRG-CRITERIA TØ Nom-Donnée-4
                                        ØN Nom-de-Fichier].
 DATA DIVISIØN.
  FILE SECTIØN.
FD     Nom-de-Fichier ⌇
                      ⌇ Description du Fichier.
```

```
A       B
WØRKING-STØRAGE SECTIØN.
        } Description de la zone Nom-Donnée-2 affectée à NØMINAL KEY.
PRØCEDURE DIVISIØN.
                                  ØPEN { INPUT  } Nom-de-Fichier
                                       { ØUTPUT }
                                       { I-Ø    }
        [START Nom-de-Fichier [INVALID KEY Ordre Impératif]]
        [ READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
                                       { AT END     }
                                       { INVALID KEY } Ordre Impératif ]
        [ { WRITE   } Nom d'Enregistrement [FRØM Nom-Donnée]
          { REWRITE }
                                  INVALID KEY Ordre Impératif ]
        CLØSE Nom-de-Fichier [WITH LØCK].
```

LES INSTRUCTIONS COBOL

SELECT Nom-de-Fichier de Data Division

ASSIGN TØ Nom-de-fichier Système

Cette clause assume toujours le même rôle de liaison entre le programme et les cartes de contrôle (J C L) du système d'exploitation.

En organisation séquentiel indexé le Nom-de-Fichier système s'écrit :

DA – I – [Nom de Fichier / Symbolique de JCL] où I désigne l'ISAM (Indexed Sequential Access Method)

Exemple :

SELECT FISEQIND ASSIGN TØ DA – I – FISEQIND

En système d'exploitation DOS, le nom de fichier Système a un format sensiblement différent :

SYSnnn – DA – [Numéro de type d'unité] – I

Exemple :

SELECT FICHIND ASSIGN TØ SYS016 – DA – 2311 – I.
SYSnnn correspond au Nom Symbolique de fichier des cartes de contrôle JCL.
Numéro de type d'unité : 2311, 2314, 3330...
I désigne le séquentiel Indexé.

$$\left[\text{RESERVE} \left\{ \begin{matrix} \text{NØ} \\ \text{Nombre entier} \end{matrix} \right\} \text{ALTERNATE AREA} \right]$$

Cette option est utilisée pour faire supprimer une des deux zones de communication (buffers) entre la mémoire centrale et le périphérique, allouées automatiquement par COBOL, lorsque le programmeur cherche à gagner de la place en mémoire, on écrit alors RESERVE NØ ALTERNATE AREA.

Mais cette option est aussi utilisée au contraire pour faire allouer des zones buffers supplémentaires afin d'augmenter les vitesses de communication avec le disque.

Par exemple : RESERVE 2 ALTERNATE AREA fournit une zone supplémentaire, soit trois zones buffers en tout.

$$\text{ACCESS MØDE IS} \left\{ \begin{matrix} \text{SEQUENTIAL} \\ \text{RANDØM} \end{matrix} \right\}$$

Cette clause indique au système d'Exploitation quelle méthode d'accès doit être utilisée : SEQUENTIAL pour QISAM ou RANDØM pour BISAM.

SEQUENTIAL est réservé au traitement séquentiel d'un fichier Séquentiel Indexé, il permet

— la création séquentielle d'un fichier,
— la lecture séquentielle d'un fichier,
— le traitement séquentiel en lecture-écriture d'un fichier avec modifications ou suppressions par HIGH-VALUE mais sans créations car les enregistrements oblitérés par HIGH-VALUE ne peuvent être obtenus en lecture séquentielle.

RANDØM est réservé au traitement aléatoire du fichier séquentiel Indexé, il permet

— la lecture aléatoire d'un fichier et dans ce cas les enregistrements oblitérés par HIGH-VALUE sont obtenus,
— la mise à jour en lecture-écriture d'un fichier avec suppressions, modifications et surtout créations.

RECØRD KEY IS Nom-Donnée-1

La RECØRD KEY ou clef d'enregistrement doit obligatoirement être présente dans tout enregistrement du fichier Séquentiel Indexé et par conséquence, on doit retrouver sa description en FILE SECTIØN.

La longueur de la RECØRD KEY peut aller jusque 255 octets et sa position doit être fixe dans l'enregistrement, même s'il existe plusieurs types d'enregistrements pour un fichier.

Rappelons enfin que la RECØRD KEY ne doit pas être placée en première position de l'enregistrement, position utilisée pour oblitérer des enregistrements par HIGH-VALUE.

[NØMINAL KEY IS Nom-Donnée-2]

La NØMINAL KEY ou clef de recherche des enregistrements doit être chargée avant toute lecture ou écriture en accès direct (RANDØM) ou pour un fichier en lecture séquentielle (SEQUENTIAL) lorsqu'on utilise l'ordre START afin de commencer le traitement ailleurs qu'au début du fichier.

Comme la RECØRD KEY, la longueur devra être de 255 octets au maximum et sa position devra être fixe en mémoire centrale, ce qui exclue une description en zone ØCCURS par exemple.

TRACK-AREA IS { Nom-Donnée-3 / Nombre Entier } CHARACTERS

Cette clause est utilisée dans le but d'améliorer les performances de traitement en accès direct RANDØM lorsqu'on désire ajouter de nouveaux enregistrements dans le fichier.

L'ajout d'un nouvel enregistrement nécessite une réorganisation de la piste, avec décalage des enregistrements suivants (si le nouveau ne prend pas la place d'un enregistrement préformaté).

La clause TRACK-AREA fournit au Système d'Exploitation une zone de travail en mémoire centrale pour améliorer la vitesse de réorganisation de la piste. On a alors réorganisation de la piste en mémoire centrale puis réécriture de la piste sur disque un peu à la manière des monotonies des tris.

TRACK-AREA doit être une zone suffisamment grande pour contenir une piste entière plus un enregistrement logique ; elle doit en outre être alignée sur des frontières de double-mot (multiple de 8 octets) et ne peut dépasser 32 760 octets.

L'option TRACK-AREA IS Nom-Donnée-3 CHARACTERS exige de déclarer la zone Nom-Donnée-3 en WØRKING-STØRAGE SECTIØN, cette zone pouvant éventuellement avoir un usage différent dans une seconde partie du programme après fermeture du fichier.

L'option TRACK-AREA IS Nombre-Entier CHARACTERS réserve une zone sans qu'il soit besoin de la décrire en WØRKING-STØRAGE SECTIØN et le système d'exploitation libère cette zone dès l'ordre CLØSE du fichier.

Notons que, après un ordre WRITE, le contenu de la NØMINAL KEY ne sera pas altéré dans le cas où la clause TRACK-AREA est utilisée alors que normalement il est impossible de retrouver la valeur initiale.

[APPLY CØRE-INDEX ØN Nom-de-Fichier]

Cette clause optionnelle est destinée à améliorer les performances de traitement des fichiers Séquentiel-Indexé en implantant en mémoire centrale la table de plus haut niveau des index.

Il en résulte un déplacement de tête de lecture en moins pour chaque lecture en accès direct RANDØM, ce qui se traduit généralement par une diminution considérable des temps de traitement et nous ne saurions trop recommander l'utilisation de cette clause lorsque la taille mémoire le permet.

Signalons que cette zone, comme la TRACK-AREA, est restituée par le système d'exploitation après l'ordre CLØSE du fichier.

[APPLY CYL-ØVERFLØW ØF Nombre TRACKS ØN Nom-de-Fichier]

Cette clause ne peut être utilisée qu'en système d'exploitation DOS car en OS on utilise les cartes JCL de contrôle.

Elle permet de réserver un Nombre de pistes sur chaque cylindre pour les zones overflow.

Si nombre = zéro, il n'y a pas réservation de pistes overflow. Si la clause est omise, COBOL réserve automatiquement une zone overflow de 20 %.

[APPLY REØRG-CRITERIA TØ Nom-donnée-4 ØN Nom-de-Fichier]

Cette option ne peut être utilisée en accès direct RANDØM qu'à condition que le paramètre ØPTCD = R ait été spécifié au préalable dans les cartes JCL de contrôle du système d'exploitation lors de la création du fichier.

Le système d'exploitation gère alors une statistique sur trois quantités binaires fournies dans la zone Nom-Donnée-4 qu'il faut déclarer en WØRKING-STØRAGE SECTIØN.

— La quantité contenue dans les deux premiers octets binaires indique le nombre de cylindres pleins dans la zone overflow.

— La quantité contenue dans les deux octets binaires suivants indique le nombre de pistes restant disponibles dans la zone overflow indépendante.

— La quantité contenue dans les quatre derniers octets indique le nombre d'ordres READ ou WRITE qui ont touché des enregistrements au-delà du premier des pistes overflow.

Ces trois quantités doivent ensuite être utilisées par le programmeur pour décider d'une organisation de fichier lorsqu'il y a trop d'enregistrements placés en zone overflow et que les temps d'exploitation s'en ressentent.

ØPEN { INPUT / ØUTPUT / I-Ø } Nom-de-Fichier

De même que nous ouvrons les fichiers dans les autres organisations, on ouvrira chaque fichier en :

— INPUT pour les lectures seules en SEQUENTIAL ou RANDØM.
— ØUTPUT pour les écritures seules en SEQUENTIAL.
— I-Ø pour les lectures et écritures en SEQUENTIAL ou RANDØM.

[START Nom-de-Fichier [INVALID KEY Ordre Impératif]]

En principe, le traitement de lecture séquentielle commence par le début du fichier, mais on peut désirer ne traiter qu'une partie de ce fichier et vouloir éviter de défiler toutes les portions inutiles qui précèdent.

On peut alors utiliser l'ordre START qui fait démarrer le traitement sur l'enregistrement dont la RECØRD KEY correspond au contenu de la NØMINAL KEY au moment de l'ordre START.

Si la RECØRD KEY n'est pas trouvée, il y a alors débranchement causé par la clause INVALID KEY.

L'ordre START peut être utilisé plusieurs fois dans le cours du programme, c'est-à-dire à chaque fois qu'on désire sauter une portion du fichier.

[START Nom-de-fichier USING KEY Nom de Clef d'Enregistrement { EQUAL TØ / = } Nom-Donnée [INVALID KEY Ordre Impératif]]

Une des innovations les plus importantes du COBOL A. N. S. est la possibilité d'utiliser une clef de groupe, c'est-à-dire une portion significative de la clef pour lancer un traitement séquentiel.

Par exemple, supposons un fichier client organisé en Sequentiel Indexé dont la clef est composée de Nom et Prénom (30 caractères). Il sera possible de lister le fichier à partir de la clef de groupe DURAN, soit seulement cinq caractères pour rechercher le bon M. DURAND ou DURANT habitant Orléans.

Dans le cas d'indicatifs composés, comportant par exemple en majeur le numéro de département, il sera possible de traiter uniquement le département du Maine-et-Loire en démarrant à la clef de groupe 49.

Nom-de-clef d'enregistrement : est le nom de la clef désignée dans la clause RECORD KEY.

Nom-Donnée : désigne une zone qui contient la clef de groupe.

Il faut prendre garde au fait que la zone Nom-Donnée doit avoir l'amplitude exacte de la clef de groupe. En effet, pour que le traitement puisse démarrer, il faut qu'on trouve effectivement égalité entre les *n* caractères de la clef de groupe et les *n* premiers caractères de gauche de la RECORD KEY. Si la zone Nom-Donnée est plus grande que la clef de groupe, le contenu sera alors complété par des caractères zéros ou blancs suivant l'émetteur, ce qui risque d'empêcher de trouver l'égalité et de déclencher une condition INVALID KEY.

```
[READ Nom-de-Fichier RECØRD [INTØ Nom-Donnée]
                            { AT END     }
                            { INVALID KEY } Ordre Impératif ]
```

Lorsqu'un fichier en Organisation Séquentiel Indexé est lu séquentiellement, on fait suivre simplement l'ordre READ de l'option AT END Ordre Impératif et il n'est pas utile d'avoir déclaré une NØMINAL KEY (sauf si on utilise conjointement l'ordre START).

En traitement aléatoire RANDØM on utilisera par contre l'option INVALID KEY conjointement à la déclaration d'une NØMINAL KEY pour la recherche des enregistrements.

Le débranchement en INVALID KEY aura lieu lorsqu'une RECØRD KEY égale au contenu de la NØMINAL KEY n'aura pas été trouvée.

```
[{ WRITE   } Nom-d'Enregistrement [FRØM Nom-Donnée]
{ REWRITE }
                                  INVALID KEY Ordre Impératif ]
```

L'ordre REWRITE est essentiellement destiné aux modifications d'enregistrements (UPDATE) pour les fichiers ouverts en lecture-écriture I-Ø.

Rappelons que les traitements séquentiels (SEQUENTIAL) permettent une modification d'enregistrements existants mais ne permettent pas la création de nouveaux enregistrements sauf s'ils remplacent des enregistrements préformatés avec HIGH-VALUE.

Pour ajouter de nouveaux enregistrements on devra travailler en accès aléatoire RANDØM et lecture-écriture I-Ø. La NØMINAL KEY est indispensable pour permettre un repérage de l'endroit où le nouvel enregistrement doit être intercalé.

CLØSE Nom-de-Fichier [WITH LØCK]

Comme toujours en COBOL, l'ordre de fermeture du fichier doit toujours être donné avant la fin du programme afin de permettre au Système d'Exploitation d'effectuer les contrôles de fin du fichier.

L'option WITH LØCK est une précaution permettant de s'assurer que le fichier ne pourra plus être réouvert intempestivement avant la fin de programme.

Tableau résumé des options utilisées en séquentiel indexé

Méthodes d'accès	ACCESS	Traitement	Ouverture ØPEN	Verbes	Fermeture CLØSE
Q. I. S. A. M.	SEQUENTIAL	Lecture	INPUT	START INVALID KEY READ [INTØ] AT END	[WITH LOCK]
		Écriture (création)	ØUTPUT	WRITE [FRØM] INVALID KEY	
		Lecture & Écriture	I–Ø	START INVALID KEY READ [INTØ] AT END REWRITE [FRØM] INVALID KEY	
B. I. S. A. M.	RANDØM	Lecture	INPUT	READ [INTØ] INVALID KEY	
		Lecture & Écriture	I–Ø	READ [INTØ] INVALID KEY WRITE [FRØM] INVALID KEY REWRITE [FRØM] INVALID KEY	

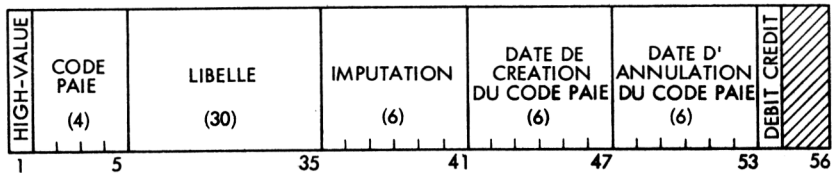
CRÉATION D'UN FICHIER EN ORGANISATION SÉQUENTIEL INDEXÉ

Rappelons que seul un traitement séquentiel (SEQUENTIAL) est possible pour créer un fichier Séquentiel indexé.

En création, seule la clef d'enregistrement RECØRD KEY est requise et le fichier émetteur devra obligatoirement être ordonné en séquence ascendante sur cet indicatif. En cas de déclassement, on a automatiquement une condition INVALID KEY car cela revient à essayer d'intercaler un nouvel enregistrement, opération qui n'est possible qu'en accès RANDØM. De plus les clefs devront être uniques, c'est-à-dire qu'on ne pourra avoir deux enregistrements dotés du même indicatif.

Afin d'éviter trop d'ajouts ultérieurs sur piste overflow, on aura tout intérêt à créer des enregistrements préformatés, oblitérés par HIGH-VALUE en première position.

Exemple : Nous désirons créer le fichier des rubriques de paie. Ce fichier a deux buts, d'une part il permet l'imputation analytique des rubriques de paie et d'autre part l'édition des libellés correspondant sur les bulletins de paie.



Ce fichier est créé à partir de cartes, de code-carte P21, ayant un format identique mais sans position HIGH-VALUE.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. CREFIIND.
REMARKS. CREATION DU FICHIER SEQUENTIEL INDEXE DES
         RUBRIQUES DE PAIE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FICARTE      ASSIGN TO   UT-S-FICARTE.
    SELECT  FISECIND     ASSIGN TO   DA-I-FISECIND
                         ACCESS IS SEQUENTIAL
                         RECORD KEY IS CODPAIE.
DATA  DIVISION.
FILE  SECTION.
FD  FICARTE      RECORDING MODE  F
                 LABEL RECORD OMITTED
                 DATA RECORD CARTE.
01  CARTE.
    02  COCART  PIC      XXX.
    02  CODEP   PIC      9(4).
    02  ENREG   PIC      X(49).
    02  FILLER  PIC      X(24).
FD  FISECIND     RECORDING MODE  F
                 BLOCK CONTAINS 60 RECORDS
                 LABEL RECORD STANDARD
                 DATA RECORD DISQUE.
01  DISQUE.
    02  FF      PIC      X.
    02  CODPAIE PIC      9(4).
    02  RESTE   PIC      X(51).
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT FICARTE   OUTPUT FISECIND.
LECTURE.
    READ FICARTE AT END GO TO FINTRA.
    IF COCART  NOT = 'P21'  GO TO ER1.
    MOVE LOW-VALUE TO FF    MOVE CODEP TO CODPAIE
    MOVE ENREG TO RESTE
    WRITE DISQUE  INVALID KEY GO TO ER2.
    GO TO LECTURE.
ER1.
    DISPLAY 'CARTE INVALIDE'  DISPLAY CARTE  GO TO LECTURE.
ER2.
    DISPLAY 'ERREUR SEQUENCE'   DISPLAY CARTE   GO TO LECTURE.
FINTRA.
    CLOSE  FICARTE  FISECIND
    DISPLAY 'FIN DE TRAVAIL'    STOP RUN.
```

On désire maintenant éditer le fichier séquentiel indexé que nous venons de créer, la NØMINAL KEY sera encore inutile puisqu'il s'agit d'un traitement séquentiel et rappelons que les enregistrements oblitérés par HIGH-VALUE ne pourront être obtenus.

```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  EDITFIS.
REMARKS. PROGRAMME  D'EDITION  DU  FICHIER
            SEQUENTIEL  INDEXE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FISECIND ASSIGN TO DA-I-FISECIND
            ACCESS IS SEQUENTIAL
            RECORD KEY IS CODEPAIE.
    SELECT  IMPRI ASSIGN TO UT-S-IMPRI.
DATA  DIVISION.
FILE  SECTION.
FD  FISECIND  RECORDING MODE F
              BLOCK CONTAINS 60 RECORDS
              LABEL RECORD STANDARD
              DATA RECORD DISQUE.
01  DISQUE.
    02  FF        PIC  X.
    02  CODEPAIE PIC  9(4).
    02  RESTE     PIC  X(51).
FD  IMPRI  RECORDING MODE F
           LABEL RECORD OMITTED
           DATA RECORD LIGNE.
01  LIGNE.
    02  SAUT     PIC  X.
    02  LIDET    PIC  X(132).
PROCEDURE  DIVISION.
DEBUT.
    OPEN INPUT FISECIND  OUTPUT IMPRI.
LECTURE.
    READ FISECIND AT END GO TO FINTRA.
    MOVE DISQUE TO LIDET
    WRITE LIGNE AFTER  1.
    GO TO LECTURE.
FINTRA.
    CLOSE FISECIND  IMPRI
    DISPLAY  'FIN  DE  TRAVAIL'
    STOP RUN.
```

MISE A JOUR EN ACCÈS SÉLECTIF D'UN FICHIER SÉQUENTIEL INDEXÉ

La mise à jour d'un fichier séquentiel Indexé peut être réalisée séquentiellement à condition d'accepter de ne pouvoir ajouter de nouveaux enregistrements ni atteindre les enregistrements oblitérés par HIGH-VALUE.

Il est donc plus logique d'utiliser les possibilités d'accès direct du séquentiel indexé et de réaliser les mises-à-jour en RANDØM.

Dans ce cas, la clef de recherche NØMINAL KEY est indispensable en plus de la clef d'enregistrement RECØRD KEY. Nous utiliserons en outre les clauses TRACK-AREA et APPLY CØRE-INDEX pour améliorer les performances de traitement.

Le principe général du traitement sera sensiblement identique à celui du traitement relatif.

A partir de l'indicatif de l'enregistrement mouvement on lit le fichier permanent séquentiel indexé :

— Si l'enregistrement du fichier permanent est trouvé, on considère deux cas :

. l'enregistrement est oblitéré par HIGH-VALUE : le mouvement doit être une création,

. l'enregistrement est valide : le mouvement doit être une modification ou suppression.

— Si l'enregistrement du fichier permanent n'est pas trouvé, le mouvement doit être une création.

Exemple : Il s'agit de réaliser la mise à jour du fichier des rubriques de paie. A partir de cartes P22, on crée, modifie ou supprime les enregistrements du fichier permanent des rubriques de paie. Les cartes P22 sont identiques aux cartes P21 de l'exemple précédent mais possèdent en colonne 80 un caractère C pour Création, M pour modification et S pour Suppression. En cas de modification, on se contente de remplacer les zones non-blanches de la carte P22.

Au fur et à mesure des mises à jour on édite :

« CARTE MISE A JOUR » et contenu de la carte modification.
« AVANT » et contenu de l'enregistrement avant modification.
« APRÈS » et contenu de l'enregistrement après modification.

EXERCICE N° 1

Nous avons étudié séparément les différentes organisations de fichiers mais il peut être utile de les associer dans le but de créer une banque de données.

Prenons l'exemple d'une banque de données PERSONNEL. La solution généralement adoptée consiste à créer un enregistrement recueillant l'ensemble des informations pour un individu ; on est alors amené à « traîner » plusieurs centaines (ou milliers) d'octets inutiles quels que soient les traitements.

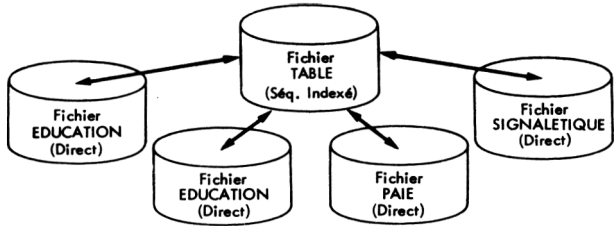
```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  RANDOMAJ.
REMARKS. MISE-A-JOUR EN ACCES DIRECT D 'UN FICHIER
         SEQUENTIEL INDEXE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FISECIND     ASSIGN TO   DA-I-FISECIND
                         ACCESS IS RANDOM
                         RECORD KEY IS CODPAIE
                         NOMINAL KEY IS CLEF
                         TRACK-AREA IS 8000.
    SELECT  FICARTE      ASSIGN TO   UT-S-FICARTE.
    SELECT  IMPRI        ASSIGN TO   UT-S-IMPRI.
I-O-CONTROL.
    APPLY  CORE-INDEX  ON  FISECIND.
DATA  DIVISION.
FILE  SECTION.
FD  FISECIND     RECORDING  F
                 BLOCK CONTAINS  60 RECORDS
                 LABEL  RECORD STANDARD
                 DATA RECORD DISQUE.
01  DISQUE.
    02  FF  PIC X.
    02  CORPS.
        04  CODPAIE PIC      9(4).
        04  LIBEL   PIC      X(30).
        04  IMPUT   PIC      X(6).
        04  DATCRE  PIC      X(6).
        04  DATFIN  PIC      X(6).
        04  DEBCRE  PIC      X.
        04  FILLER  PIC      XX.
FD  FICARTE RECORDING  F
            LABEL RECORD OMITTED
            DATA RECORD CARTE.
01  CARTE.
    02  COCART  PIC      XXX.
    02  CACORP.
        04  CODEP   PIC      9(4).
        04  CARLIB  PIC      X(30).
        04  CARIMP  PIC      X(6).
        04  CARCRE  PIC      X(6).
        04  CARFIN  PIC      X(6).
        04  CARDB   PIC      X.
        04  FILLER  PIC      X(23).
        04  CODMAJ  PIC      X.
FD  IMPRI    RECORDING MODE F
             LABEL RECORD OMITTED
             DATA RECORD LIGNE.
01  LIGNE   PIC   X(133).
WORKING-STORAGE  SECTION.
77  CLEF   PIC  9(4).
01  LI1.
    02  TITRE   PIC     X(17).
    02  ENR1    PIC     X(56).
    02  FILLER  PIC     X(60)  VALUE SPACE.
01  LI2.
```

```
    02  FILLER  PIC X(23) VALUE 'CARTE MISE A JOUR = '.
    02  ENR2    PIC X(80).
    02  FILLER  PIC X(30)  VALUE  SPACE.
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT     FICARTE  OUTPUT IMPRI  I-O  FISECIND.
LECTURE.
    READ FICARTE AT END CLOSE FISECIND FICARTE IMPRI  STOP RUN.
    MOVE CODEP TO CLEF
    MOVE CARTE TO ENR2  WRITE LIGNE FROM LI2
    READ FISECIND  INVALID KEY  GO TO INCONNU.
    MOVE DISQUE TO ENR1      MOVE 'AVANT' TO TITRE
    WRITE LIGNE FROM LI1
    IF  FF = HIGH-VALUE  GO TO INCONNU.
    IF CODMAJ = 'C'    GO TO ER1.
    IF CODMAJ = 'M'    GO TO MODIFICATION.
    IF CODMAJ = 'S'    GO TO SUPPRESSION.
    MOVE 'ERREUR2' TO TITRE
    MOVE 'CODE MAJ INCONNU' TO ENR1
    WRITE LIGNE FROM LI1     GO TO LECTURE.
ER1.
    MOVE 'ERREUR1' TO TITRE
    MOVE 'MAJ A TORT'  TO  ENR1.
    WRITE LIGNE FROM LI1     GO TO LECTURE.
INCONNU.
    IF  CODMAJ NOT = 'C'  GO TO ER1.
CREATION.
    MOVE LOW-VALUE  TO FF
    MOVE  CACORP TO CORPS
    WRITE DISQUE  INVALID KEY GO TO ER3.
    GO TO ECRIT.
SUPPRESSION.
    MOVE HIGH-VALUE TO FF
    REWRITE DISQUE  INVALID KEY GO TO ER3.
    MOVE 'APRES' TO TITRE    MOVE DISQUE TO ENR1.
ECRIT.
    WRITE LIGNE FROM LI1     GO TO LECTURE.
ER3.
    MOVE  'ERREUR3'  TO TITRE
    MOVE  'ECRITURE IMPOSSIBLE'  TO ENR1
    GO TO ECRIT.
MODIFICATION.
    IF  CARLIB  NOT = SPACE MOVE     CARLIB  TO  LIBEL.
    IF  CARIMP  NOT = SPACE MOVE     CARIMP  TO  IMPUT.
    IF  CARCRE  NOT = SPACE MOVE     CARCRE  TO  DATCRE.
    IF  CARFIN  NOT = SPACE MOVE     CARFIN  TO  DATFIN.
    IF  CARDB   NOT = SPACE MOVE     CARDB   TO  DEBCRE.
    REWRITE  DISQUE  INVALID KEY GO TO ER3.
    MOVE  'APRES' TO TITRE  MOVE DISQUE TO ENR1
    GO TO ECRIT.
```

Il est préférable de regrouper les informations suivant leur nature :

— informations signalétiques,
— informations de paie,
— informations médico-sociales,
— informations d'éducation, etc.

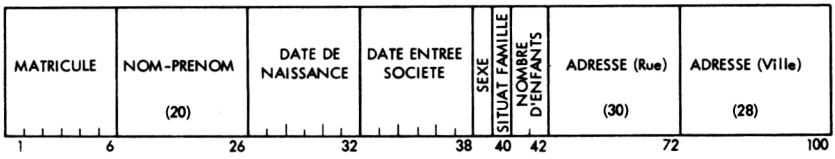
et de constituer autant de fichiers en organisation directe, reliés par un fichier table en séquentiel indexé.



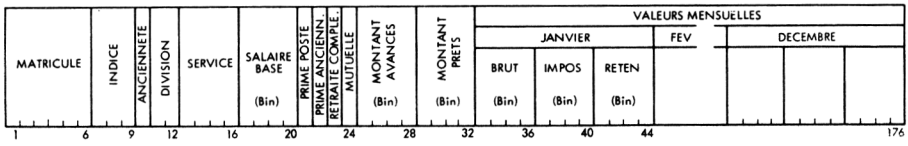
De cette manière, pour les traitements séquentiels de paie par exemple, il est possible de n'utiliser que le fichier PAIE avec les performances de l'organisation directe. Pour les traitements en accès direct, on passe par l'intermédiaire du fichier table qui fournit la clef, donc l'adresse disque, des différents enregistrements correspondant à un employé.

A titre d'exercice, nous nous proposons de créer le fichier signalétique, le fichier paie ainsi que le fichier table dans un même programme.

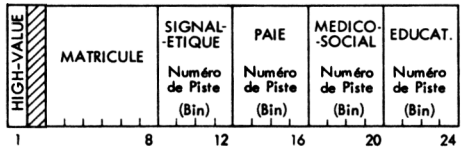
L'enregistrement signalétique a le format suivant :



L'enregistrement salaire a le format suivant :

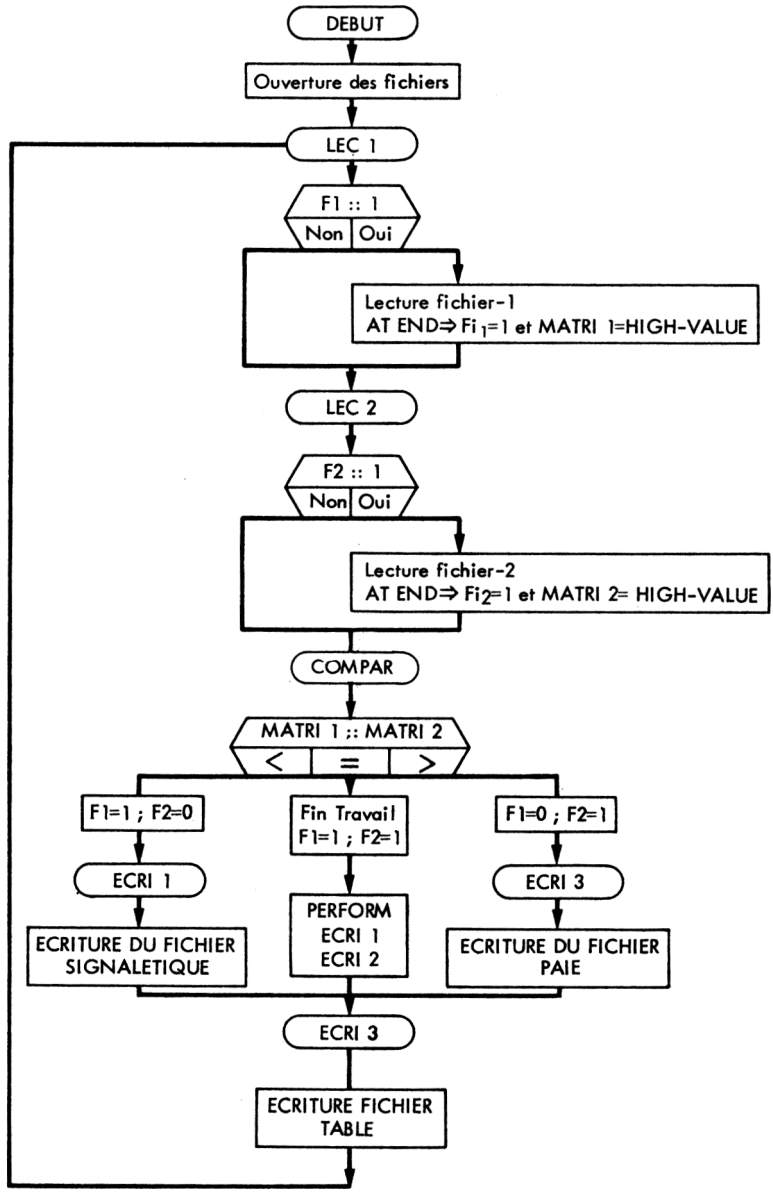


L'enregistrement table a le format suivant :



Les fichiers sont créés à partir de bandes comportant environ 16 000 enregistrements chacune et on suppose que les nouveaux matricules sont ouverts à la suite des anciens.

CORRIGÉ DE L'EXERCICE 1



F1 = Variable d'occurence du fichier-1 (Valeur initiale « 1 »)
F2 = Variable d'occurence du fichier-1 (Valeur initiale « 1 »)
Fi1 = Variable de fin de fichier-1 (Valeur initiale « 0 »)
Fi1 = Variable de fin de fichier-1 (Valeur initiale « 0 »)

CHAPITRE IV

VSAM

Avec les systèmes d'exploitation supportant la mémoire virtuelle sont apparues les nouvelles méthodes d'accès VSAM (Virtual Storage Access Method) destinées à remplacer progressivement les méthodes d'accès antérieures.

VSAM recouvre trois organisations :

— *Organisation consécutive* ESDS (Entry Sequenced Data Set)

Organisation dans laquelle les données sont enregistrées dans l'ordre d'arrivée, sans espaces libres, et accessibles par adressage relatif (compté en nombre d'octets et non pas en enregistrements).

On ne peut supprimer des enregistrements mais on peut en ajouter des nouveaux en fin de fichier.

— *Organisation indexée* KSDS (Keyed Sequenced Data Set)

Organisation dans laquelle les données sont enregistrées en séquence sur une zone clé, avec espaces libres, et accessibles par l'indexage construit avec la clé par adressage relatif (compté aussi en nombre d'octets).

On peut modifier, ajouter ou supprimer un quelconque enregistrement.

— *Organisation relative* RRDS (Relative Record Data Set)

Organisation dans laquelle les enregistrements sont de longueur fixe et ordonnés en séquence sur un numéro d'ordre qui désigne leur position relative dans le fichier.

On peut modifier des enregistrements, en insérer de nouveaux dans les places vacantes et en ajouter à la fin du fichier.

VSAM autorise en plus l'utilisation d'index secondaires, index qui permettent d'accéder directement aux enregistrements indexés KSDS sur une clé différente de celle qui a permis de générer le fichier.

La principale originalité des méthodes d'accès VSAM réside dans la structure logique des fichiers, composés de « CONTROL AREA » et « CONTROL INTERVAL », et surtout par l'apparition d'un langage spécialisé de contrôle, l'AMS (Access Method Services).

LE LANGAGE AMS

Toutes les informations caractéristiques d'un fichier VSAM doivent être inscrites dans un *catalogue VSAM* — Catalogue Maître (MASTER)

uniquement en système DOS/VS — Catalogue Maître ou utilisateur (USER) en système OS/VS.

Ces catalogues, indispensables et constamment disponibles, éviteront de répéter les caractéristiques de fichier VSAM avec tous les programmes qui les utiliseront et permettront une optimisation automatique de la gestion de ces fichiers par VSAM.

C'est par l'intermédiaire du langage AMS et son traducteur IDCAMS qu'on peut créer un catalogue VSAM, y inscrire des descriptions de fichiers et même éventuellement manipuler des fichiers.

Notre but n'est pas l'étude du langage A. M. S., mais nous indiquerons les procédures nécessaires à la création d'un fichier VSAM avant d'étudier les instructions COBOL de gestion des organisations ESDS et KSDS.

Ce langage est plus proche du langage JCL de contrôle d'exploitation et de PL/1 que de COBOL. Il comporte des verbes commandes, des paramètres à mots clés et des identificateurs (noms de fichiers) qui suivent les règles d'écriture du langage JCL.

Ajoutons pour plus de précisions que :

- Le format est libre entre les colonnes 2 à 72.
- Un point-virgule termine chaque instruction (il est toutefois permis de l'oublier).
- On ne peut mettre plus d'une instruction par ligne.
- Lorsqu'une instruction ne tient pas tout entière sur une ligne, il faut mettre un tiret (−), caractère de continuation précédé d'un caractère blanc.
- Lorsqu'un verbe-commande ou nom-symbolique ne peut tenir tout entier, il faut utiliser le caractère +, caractère de continuation d'identificateur.
- On peut insérer des commentaires, délimités comme en PL/1 par les couples de caractères /* au début et */ en fin.

STRUCTURE LOGIQUE DES FICHIERS VSAM

CONTROL AREA : un fichier VSAM est composé d'un nombre entier de zones, appelées CONTROL AREA, dont la taille sera imposée par VSAM (en général un cylindre de disque au maximum).

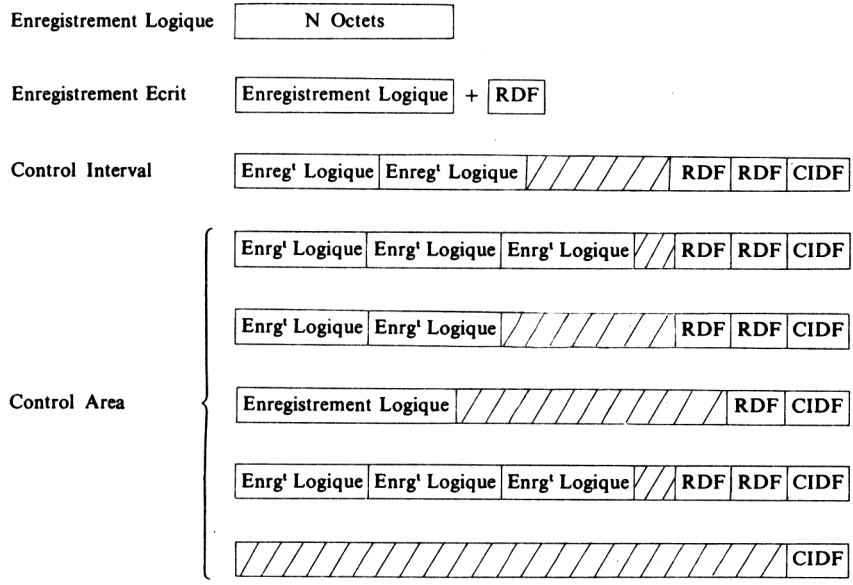
Une zone CONTROL AREA comporte un nombre entier de zones appelées CONTROL INTERVAL.

CONTROL INTERVAL : un control interval contient plusieurs blocs physiques d'enregistrements logiques. La taille du control interval, en principe déterminée par VSAM (mais le programmeur peut l'imposer), dépend de la taille maximum des enregistrements, de la taille minimum des zones de communications (buffers) d'Entrées-Sorties et des caracté-

ristiques de l'unité physique. En fait, un control interval a pour taille 512, 1 024, 2 048, 4 096, etc., octets.

Les caractéristiques de remplissage du control interval sont contenues dans un CIDF (Control Interval Definition Field) en fin de zone (2 octets désignent le premier octet libre et deux autres octets pour la longueur d'espace libre).

A chaque enregistrement logique contenu dans le control interval est associé un RDF (Record Definition Field) de 3 octets qui le définit. En format variable, chaque RDF contient la longueur de l'enregistrement associé ; par contre, en format fixe, il n'y a que deux RDF : le premier donne la longueur commune des enregistrements et le second donne le nombre d'enregistrements de même longueur.



RDF = Record Definition Field.
CIDF = Control Interval Definition Field.

Structure logique d'un fichier VSAM.

FICHIER INDEX ET CLUSTER

Nous avons dit que l'organisation KSDS permet d'accéder aux enregistrements par indexages.

Ceci est réalisé par des fichiers d'index indépendants, effectivement structurés comme les autres fichiers VSAM.

On introduit alors la notion de CLUSTER pour relier fichiers INDEX et fichiers KSDS contenant les données.

Remarque : on pourra aussi parler de CLUSTER pour un fichier ESDS, mais le fichier index étant alors nul, le CLUSTER ne désigne que le fichier des données.

INDEX SECONDAIRES

Il est possible de définir des index secondaires pour des fichiers KSDS, c'est-à-dire de permettre l'accès aux données suivant des classements différents de l'ordre initial.

Un index secondaire suppose un espace supplémentaire (place disque) pour loger un cluster ALTERNATEINDEX contenant les tables de l'index secondaire.

Il sera aussi nécessaire de relier logiquement le cluster de base et le cluster secondaire pour définir la voie d'accès secondaire aux données appelée PATH.

Un chapitre particulier traite des Index Secondaires.

CONFIDENTIALITÉ. SÉCURITÉ D'ACCÈS

Il est possible de protéger l'accès à un catalogue ou fichier en affectant des « mots de passe » (1 à 8 caractères alpha-numériques) qu'il sera nécessaire de fournir ensuite systématiquement à chaque tentative d'accès au catalogue ou fichier.

Les différents niveaux de protection et mots de passe correspondants sont indiqués au paragraphe suivant : « CRÉATION D'UN CATALOGUE ».

CRÉATION D'UN CATALOGUE

Avant de pouvoir utiliser VSAM, il est nécessaire de posséder un catalogue maître (MASTERCATALOG) et, le cas échéant en OS, un catalogue utilisateur (USERCATALOG).

Le format général de définition d'un catalogue est le suivant :

```
DEFINE { MASTERCATALØG }                                          –
       { USERCATALØG    }
       (NAME (Nom du catalogue)                                   –
       ┌──────────────────────────────────────────────────────┐   –
       │ FILE (Nom de Fichier externe DDNAME)                 │
       │ { VØLUME } (n° de Volume)                            │
       │ { VØL    }                                           │
       │   { { TRACKS } (primaires [,b secondaires])          │
       │   { { TRK    }                                       │
       │   { { CYLINDERS } (primaires [,b secondaires])       │  – Paramètres
       │   { { CYL       }                                    │    d'Allocation
       │   { { RECØRDS } (primaires [,b secondaires])         │
       │   { { REC     }                                      │
       └──────────────────────────────────────────────────────┘
```

[MØDEL (Nom de Catalogue utilisateur-2
[/mot de passe-1])] * –

[{ BUFFERSPACE / BUFSPC / BUFSP } (dimension)] –

[{ WRITECHECK / WCK }]
[{ NØWRITECHECK / NWCK }] –
[{ MASTERPW / MRPW } (mot de passe-2)] –
[{ CØNTRØLPW / CTLPW } (mot de passe-3)] –
[{ UPDATEPW / UPDPW } (mot de passe-4)] – Paramètres de Protection
[{ READPW / RDPW } (mot de passe-5)] –
[CØDE (Nom-Codé)] –
[{ ATTEMPTS / ATT } (nombre)] –
[{ AUTHØRIZATIØN / AUTH }
(Adresse symbolique point d'entrée de Module [*b* Chaîne de Caractères])] –
[TØ (date AAJJJ) / FØR (Nombre de Jours)] –
[ØWNER (Nom du propriétaire)])–

[DATA (paramètres optionnels)] –
[INDEX (paramètres optionnels)] –
[CATALØG (Nom du catalogue MASTER/
mot de passe-6)] * –

Exemple : En système OS

```
// CRECAT   JØB~~~~~~~~~~~~~~~~
// STEP01   EXEC PGM=IDCAMS
// DDCAT    DD   UNIT=3330, VØL=SER=111111, DISP=ØLD
// SYSPRINT DD   SYSØUT=A
// SYSIN    DD   *
```

* Pour catalogue utilisateur uniquement.

```
DEFINE MASTERCATALØG                          –
       (NAME (MASTCAT)                        –
        FILE (DDCAT)                          –
        VØLUME (111111) CYLINDERS (10,2)      –
        MASTERPW (SESAME)                     –
        UPDATEPW (ECRITURE));
/*
```

Les paramètres d'allocation :

FILE (Nom de Fichier externe DDNAME).

Le paramètre FILE établit un lien entre le nom-externe indiqué entre parenthèses et le nom de fichier DDNAME désigné dans les cartes JCL de contrôle d'exploitation. Dans l'exemple précédent il s'agit de DDCAT.

{ VØLUME / VØL } (n° de volume) désigne le numéro de volume disque sur lequel on désire placer le catalogue. On prendra garde qu'on ne peut définir qu'un seul catalogue par volume disque. La dimension de l'espace à réserver est indiquée en:

TRACKS = pistes
CYLINDERS = cylindres
RECØRDS = enregistrements logiques.

Paramètres particuliers :

[MØDEL (Nom de catalogue utilisateur-2 [/mot de passe-1])]

Ce paramètre n'est valable que pour un catalogue utilisateur. Il désigne un autre catalogue utilisateur, déjà enregistré, dont on désire recopier les paramètres, excepté les paramètres d'allocation.

[{ BUFFERSPACE / BUFSPC / BUFSP } (dimension)] désigne la taille minimum du buffer qui sera allouée à VSAM pour accéder au catalogue.

Les valeurs autorisées vont de 3 072 octets (valeur par défaut) à 8 192 octets, par multiples de 512 comme les control-interval.

Paramètres de protection :

[{ WRITECHECK / WCK } / { NØWRITECHECK / NWCK }] permet d'indiquer si on désire un contrôle systématique des informations après écriture. Par défaut, il n'y a pas de contrôle (NWCK).

[{MASTERPW / MRPW} (mot de passe-2)] désigne le mot de passe de plus haut niveau (mot de passe maître) permettant tout accès et modifications sur un catalogue.

[{CØNTRØLPW / CTLPW} (mot de passe-3)] désigne le mot de passe qui protège le catalogue contre des lectures ou écritures au niveau CI (Control Interval).

[{UPDATEPW / UPDPW} (mot de passe-4)] désigne le mot de passe qui protège le catalogue contre toute écriture au niveau enregistrement logique.

[{READPW / RDPW} (mot de passe-5)] désigne le mot de passe qui protège le catalogue contre toute lecture.

[CØDE (Nom-Codé)] plutôt que de désigner un catalogue par son nom plus un mot de passe, on peut utiliser directement un Nom-Codé.

Exemple : CØDE (TØUCIF).

[{ATTEMPTS / ATT} (Nombre)] désigne le nombre de tentatives autorisé pour la fourniture d'un mot de passe valide.

Par défaut on autorise deux tentatives. Le nombre peut aller de 0 à 7.

[{AUTHØRIZATIØN / AUTH} (Adresse symbolique du point d'entrée de module [/ chaîne de caractères])]

Ce paramètre est à utiliser lorsqu'on désire que des contrôles supplémentaires soient effectués par un module désigné par l'adresse symbolique du point d'entrée (Nom-Externe).

La chaîne de caractères (255 caractères maxi) représente les informations qu'il faut fournir au module de contrôle.

[TØ (date AAJJJ) / FØR (Nombre de Jours)] permet de spécifier une date de rétention, au-delà de laquelle le catalogue sera périmé.

— avec l'option TØ, il faut spécifier une date limite sous la forme AAJJJ (Année, jour).
— avec l'option FØR, on indique un nombre de jours entre 0 et 1830,
— par défaut le nombre de jours est égal à zéro, mais on ne pourra détruire le fichier qu'à condition qu'il soit vide (voir la commande DELETE).

[ØWNER (Nom de propriétaire)] permet d'enregistrer le nom du propriétaire du catalogue.

Autres paramètres :

[DATA (paramètres optionnels)]

Cette option signifie que l'espace créé pour le catalogue est « partageable » avec d'autres fichiers.

Les paramètres optionnels spécifient alors la place réservée au catalogue lui-même, le reste étant disponible pour l'enregistrement de clusters.

Les paramètres optionnels sont : TRACKS, CYLINDERS, RECØRDS, WRITECHECK, BUFFERSPACE.

Lorsque cette option n'est pas indiquée, l'espace est considéré « unique ».

[INDEX (paramètres optionnels)]

Ce paramètre est semblable au précédent. Les espaces indiqués dans DATA et INDEX sont soustraits de l'espace attribué au catalogue. Le reste est alors laissé disponible pour l'enregistrement de clusters.

Les paramètres optionnels sont : TRACKS, CYLINDERS, RECØRDS, WRITECHECK.

[CATALØG (Nom du catalogue MASTER/mot de passe)]

Ce paramètre n'est utile que dans le cas d'un catalogue utilisateur subordonné à un catalogue MASTER doté d'un mot de passe.

CRÉATION DE FICHIERS VSAM

VSAM réalise lui-même la gestion de la place nécessaire aux fichiers sur disques. C'est pourquoi il faut allouer à VSAM des espaces (SPACE) qui pourront ne contenir qu'un seul fichier (espace dit unique) ou plusieurs fichiers (espace dit partageable).

Créer un fichier impose donc d'allouer un nouvel espace ou de partager un espace déjà attribué pour que VSAM inscrive toutes les caractéristiques de ce fichier dans le catalogue.

Ceci n'est encore possible qu'avec le langage AMS et son traducteur IDCAMS.

Définition d'un espace VSAM :

Le format général de définition d'un espace (donc d'un fichier) VSAM est le suivant :

```
DEFINE SPACE
        (FILE (Nom de fichier externe DDNAME)
         VØLUMES (n° de volume-1 [/ n° de volume-2] ...)
        { TRACKS (primaires [/ secondaires])                          }
        { CYLINDERS (primaires [/ secondaires])                       }
        { RECØRDS (primaires [/ secondaires] / RECØRDSIZE             }
        {                                  (moyenne / maximum))       }
        { CANDIDATE                                                   }
        [CATALØG (nom du catalogue [/mot de passe]
                                   [nom du catalogue DDNAME])] ;
```

Exemple : en système OS.

```
// CRESPAS  JØB~~~~~~~~~~~~~~~~
// STEPCAT  DD DSN = VSAMCAT, UNIT = 3330,
            VØL = SER = 111111, DISP = SHR
// STEP01   EXEC PGM = IDCAMS
// VSAMSPA  DD UNIT = 3330, VØL = SER = 222222, DISP = ØLD
// SYSPRINT DD SYSØUT = A
// SYSIN    DD *
    DEFINE SPACE                                    -
           (FILE (VSAMSPA)                          -
            VØL (222222) CYL (10,1))                -
            CATALØG (VSAMCAT/SECRET);
/*
```

FILE (Nom de fichier externe DDNAME)

Le paramètre FILE établit un lien entre le nom externe indiqué entre parenthèses et le nom de fichier DDNAME désigné dans les cartes JCL de contrôle d'exploitation. Dans l'exemple précédent il s'agit de VSAMSPA qu'on déclare DISP = ØLD ou SHR pour éviter les contrôles du système d'exploitation qui en principe exigent une déclaration d'espace SPACE = ~ pour tout fichier NEW.

VØLUMES (n° de volume-1 [, n° de volume-2] -----)

Désigne le (ou les) numéro de volume disque sur lequel on devra réserver l'espace. Lorsqu'on indique plusieurs n^{os} de volumes, VSAM réservera des espaces identiques sur chacun d'eux.

La dimension de l'espace à réserver est indiquée en :

TRACKS = pistes
CYLINDERS = cylindres
RECØRDS = enregistrements logiques dont il faut indiquer la dimension moyenne et maximum.
CANDIDATE = réservation potentielle, c'est-à-dire que l'espace n'est pas réellement alloué mais que l'espace vide est provisoirement enregistré au catalogue.

$$\left[\text{CATALØG}\left(\text{nom du catalogue [/mot de passe]}\left[\begin{array}{c}\text{nom du fichier}\\ \text{externe DDNAME}\end{array}\right]\right)\right]$$

Permet de désigner dans quel catalogue l'espace VSAM devra être enregistré.

En système DOS/VS il s'agit obligatoirement du catalogue MASTER, donc l'information n'a guère d'utilité, sauf si on désigne un mot de passe pour interdire la mise à jour du catalogue à n'importe qui.

En OS/VS, par contre, on pourra désigner un nom de catalogue utilisateur et le nom externe correspondant, désigné dans les cartes JCL de contrôle d'exploitation.

Ne pas oublier une carte // STEPCAT DD~ ou bien // JØBCAT DD~ pour permettre au Système d'Exploitation de localiser le catalogue si celui-ci n'est pas le Master catalogue.

* Système OS uniquement.

CRÉATION D'UN CLUSTER

Les catalogues et espaces VSAM étant définis, il faut maintenant définir un CLUSTER pour relier logiquement les parties données et index d'un fichier indexé KSDS.

Rappelons que cette notion est étendue aux fichiers consécutifs ESDS dont la partie index est considérée nulle.

La définition du cluster génère un poste au catalogue.

Le format général de définition d'un cluster est le suivant :

```
DEFINE CLUSTER
        (NAME (Nom du cluster)
        [ { NØNINDEXED }                                                  ]
        [ { NIXD       }                                                  ]
        [   NUMBERED                                                      ]
        [ { INDEXED } [REPLICATE   ]                                      ]
        [ { IXD     } [NØREPLICATE ]                                      ]
        [                         [IMBED   ] [KEYS (Longueur, position)]  ]
        [                         [NØIMBED ]                              ]

        [ VØLUMES                 ]
        [ TRACKS                  ]  comme
        [ CYLINDERS               ]  définition
        [ RECØRDS                 ]  de cluster
        [ [ { ØRDERED } ]                                                 ]
        [ [ { ØRD     } ]                                                 ]
        [ [ { UNØRDERED } ]                                               ]
        [ [ { UNØRD     } ]                                               ]
        [ [REUSE   ]                                                      ]
        [ [NØREUSE ]                                                      ]
        [ [SPEED   ]                                                      ]
        [ [RECØVERY ]                                                     ]
        [ [ { UNIQUE } FILE (Nom externe DDNAME)                          ]
        [ [ { UNQ    }                                                    ]
        [ [ { SUBALLØCATIØN }                                             ]
        [ [ { SUBAL         }                                             ]
        [ [ { FREESPACE } (% par CI [b % par CA])                         ]
        [ [ { FSPC      }                                                 ]
        [ [ { CØNTRØLINTERVALSIZE }                                       ]
        [ [ { CNVSZ               } (dimension)                           ]
        [ [ { CISZ                }                                       ]
        [ [ { KEYRANGES } ((clé inférieure b clé supérieure))             ]
        [ [ { KRNG      }                                                 ]
```

Paramètres d'Allocation

[MØDEL~~]] comme définition
[BUFFERSPACE]] de catalogue

[WRITECHECK~~~~~]]
[NØWRITECHECK~~~]]
[MASTERPW ~~~~~~~]]
[CØNTRØLPW~~~~~~]]
[UPDATEPW ~~~~~~~]] comme
[READPW~~~~~~~~~~]] définition
[CØDE ~~~~~~~~~~~]] de
[ATTEMPTS~~~~~~~~]] catalogue
[AUTHØRIZATIØN~~~]]
[TØ~~~~]]
[FØR~~~]]
[ØWNER~~~]]

[{ SHAREØPTIØNS } (entre régions [*b* entre systèmes])]
[{ SHR }]

[ERASE]
[NØERASE]

Paramètres de Protection

)

[DATA~~~~~~~]]
[INDEX~~~~~~]] comme définition de catalogue
[CATALØG~~~]]

Exemple : en système OS/VS.

```
// DEFCLUST JØB~~~~~~~~~~~~~~~~
// STEP01    EXEC PGM = IDCAMS
// STEPCAT   DD DSN = MYCATAL, UNIT = 3330,
             VØL = SER = 444444, DISP = SHR
// SYSPRINT  DD   SYSØUT = A
// SYSIN     DD   *
     DEFINE CLUSTER                    -
            (NAME (VSINDEX)            -
             VØL (111111,222222)       -
             ØRDERED                   -
             CYL (100,2)               -
             RECØRDSIZE (256,256)      -
             INDEXED KEYS (8,3))       -
       CAT (MYCATAL/ABIBI);
/*
```

{ NØNINDEXED } signifie qu'il s'agit d'un cluster ESDS.
{ NIXD }

NUMBERED désigne un fichier d'organisation relative RRDS pour lequel les enregistrements sont désignés par un numéro d'ordre.

{ INDEXED } désigne un cluster KSDS. C'est le mode pris par défaut
{ IXD } et dans ce cas on peut spécifier les paramètres suivants :

REPLICATE signifie que l'index sera répété autant de fois que possible sur la piste pour augmenter les performances d'accès.

NØREPLICATE est l'option par défaut, c'est-à-dire pas de répétitions.

IMBED le séquence set est placé et répété plusieurs fois sur une piste précédant les données pour améliorer les performances d'accès (voir plus loin les fichiers indexés KSDS).

NØIMBED est l'option contraire prise par défaut.

KEYS (longueur, position) définit en nombre d'octets la longueur et la position (comptée à partir de zéro) de la clé dans l'enregistrement).

{ØRDERED / ØRD} oblige VSAM à utiliser les volumes dans l'ordre indiqué.

[REUSE / NØREUSE] REUSE signifie que le cluster doit être considéré vide à chaque utilisation, à condition que dans le programme COBOL le fichier soit ouvert en mode ØUTPUT.

REUSE sera donc utile pour un fichier temporaire de travail. Plus généralement REUSE évitera d'avoir recours à la commande DELETE du langage AMS à chaque fois qu'on veut restaurer un fichier. En effet, un fichier se charge en mode ØUTPUT (en COBOL) mais lorsque le fichier existe déjà les données sont ajoutées à la suite. Pour restaurer un fichier il faut donc au préalable utiliser la commande DELETE du langage AMS, ou bien il faut avoir déclaré le cluster REUSE dans la commande DEFINE CLUSTER. REUSE ne peut être spécifié pour un fichier de base devant supporter des index secondaires.

{SPEED / RECØVERY} RECØVERY signifie que VSAM doit préformater l'espace alloué au cluster avant chargement pour faciliter les procédures de reprises en cas d'incident.
SPEED permet d'aller plus vite car élimine le formatage.

{UNIQUE / UNQ} on désire créer un cluster unique dans un espace réservé à lui seul. Dans ce cas, en même temps qu'on définit le cluster, on crée un espace, ce qui impose de fournir le paramètre FILE.

{SUBALLØCATIØN / SUBAL} signifie, au contraire du cas précédent, que le cluster doit être alloué sur un espace existant.

{FREESPACE / FSPC} (% par CI [*b* % par CA]) permet d'indiquer les pourcentages (de 0 à 100) d'espace à laisser libre par control interval (CI) et control area (CA) pour un fichier KSDS. Cet espace libre sera utilisé ultérieurement pour des ajouts d'enregistrements.

{ CØNTRØLINTERVALSIZE / CNVSZ / CISZ } (dimension) permet d'imposer une dimension des control interval (CI) en nombre d'octets.

{ KEYRANGES / KRNG } ((clé inférieure *b* clé supérieure)) (clé inférieure *b* clé supérieure)

Pour un fichier KSDS, on peut imposer la répartition des enregistrements sur différents volumes en indiquant des séries de clés.

Exemple :

```
DEFINE  CLUSTER
        (NAME    (VSINDEX)
         VOL     (111111,222222,333333)
         KNRG ((000 250) (251 500) (501 750))
         ØRDERED
```

{ SHAREØPTIØNS / SHR } (entre régions [*b* entre systèmes]).

Ce paramètre indique les possibilités d'accès simultanés au cluster à partir de plusieurs régions ou même de plusieurs systèmes.

On écrit alors les valeurs suivantes :

1 — Autorise plusieurs lectures ou une écriture
2 — Autorise plusieurs lectures et une écriture
3 — Toutes opérations sont autorisées sans contrôle de VSAM
4 — Toutes opérations sont autorisées avec contrôle de VSAM

[ERASE / NØERASE] En cas de destruction de fichiers confidentiels par la fonction DELETE (donc intentionnellement), ERASE signifie que les enregistrements devront être écrasés par des zéros binaires, alors qu'avec NØERASE le cluster est simplement supprimé du catalogue.

AUTRES COMMANDES AMS

Il existe d'autres commandes AMS. Nous nous bornerons à donner brièvement le format général des commandes ALTER, DELETE, REPRØ, LISTCAT, VERIFY et PRINT.

— La commande ALTER permet de modifier les informations enregistrées d'un poste catalogue, espace ou cluster.

— La commande DELETE supprime un poste dans un catalogue.

— La commande REPRØ permet de créer un fichier VSAM à partir d'un autre fichier.

— La commande LISTCAT liste le contenu d'un catalogue VSAM.
— La commande VERIFY permet de vérifier qu'un fichier VSAM est bien opérationnel.
— La commande PRINT permet d'éditer le contenu d'un fichier VSAM.

ALTER Nom du poste à modifier [/mot de passe]

[CATALØG (Nom du catalogue [/mot de passe] [DDNAME])]
[NEWNAME [Nouveau Nom de poste)]
[FILE (Nom de fichier externe DDNAME)]
[{ ADDVØLUMES / AVØL } (volumes à ajouter)]
[{ REMØVEVØLUMES / RVØL } (volumes inutilisés à retirer de la liste)]
[FREESPACE (nouveaux pourcentages)]
[{ NULLIFY / NULL } [paramètres de protection à annuler]]
[Paramètres de protection à créer];

Exemple :

```
// MØDIVSAM JØB~~~~~~~~~~~~~~~~
// STEP01   EXEC PGM = IDCAMS
// SYSPRINT DD   SYSØUT = A
// SYSIN    DD *
       ALTER VSDØNNEE          -
             FREESPACE (20,20);
/*
```

DELETE (Nom du poste à supprimer [/mot de passe])

[CLUSTER / SPACE / MASTERCATALØG / USERCATALØG / NØNVSAM / ALTERNATEINDEX / PATH] } Type de poste à supprimer

[CATALØG (Nom du catalogue [/mot de passe] [DDNAME])]
[FILE (Nom de fichier externe DDNAME)]
[PURGE / NØPURGE]
[ERASE / NØERASE]
[SCRATCH / NØSCRATCH];

Exemple :

```
// SUPPRESS JØB
// STEP01    EXEC PGM = IDCAMS
// SYSPRINT  DD   SYSØUT = A
// SYSIN     DD   *
    DELETE   (VSCLUS)                    -
             CLUSTER                     -
             CATALØG (VSAMCAT/ABIBI)     -
             PURGE;
/*
```

ALTERNATEINDEX et PATH permettent de supprimer un index secondaire ou un chemin associé.

PURGE indique qu'il ne faut pas tenir compte de la date de rétention. Avec NØPURGE (pris par défaut) la commande sera refusée si la date de rétention n'est pas atteinte.

NØSCRATCH s'applique aux fichiers non VSAM qu'on demande de ne pas effacer du catalogue classique VTOC. Avec SCRATCH (pris par défaut) le fichier sera effacé de la VTOC en même temps que du catalogue VSAM.

CHARGEMENT DE FICHIERS VSAM
CONVERSION DE FICHIERS A VSAM

La commande REPRØ permet de convertir automatiquement un fichier quelconque en fichier VSAM ; donc en particulier permet de convertir les fichiers séquentiels indexés.

Le format général de la commande REPRØ est le suivant :

REPRØ

INFILE (Nom DDNAME du Fichier en Entrée [/mot de passe] [ENVIRØNMENT (DUMMY / liste de paramètres)])

ØUTFILE (Nom DDNAME du Fichier en Sortie [/mot de passe])

{ FRØMKEY (clé-1) / FRØMADDRESS (adress-1) / FRØMNUMBER (Nombre-1) / SKIP (nombre-1) }

{ TØKEY (clé-2) / TØADDRESS (adresse-2) / TØNUMBER (nombre-2) / CØUNT (nombre-2) } ;

Exemple : OS :

```
// CØNVERS JØB~~~~~~~~~~~~~~~
// STEP01   EXEC PGM = IDCAMS
// FISAM    DD   DSN  = FICHISAM, DISP=ØLD, DCB=DSØRG=IS
// FIVSAM   DD   DSN  = FIVSAM, DISP = ØLD
// SYSPRINT DD   SYSØUT = A
// SYSIN    DD   *
         REPRØ  INFILE   (FISAM)    -
                ØUTFILE (FIVSAM) ;
/*
```

ENVIRØNMENT (DUMMY) est réservé en OS aux fichiers ISAM lorsqu'on désire que les enregistrements oblitérés par FF soient aussi recopiés sur le fichier VSAM en sortie.

ENVIRØNMENT (liste de paramètres) est destiné en DOS à fournir les caractéristiques de fichier en entrée :

RECØRDFØRMAT (code) avec code = F, FB, V, VB, S, SB, U
BLØCKSIZE (dimension)
RECØRDSIZE (dimension)
HINDEXDEVICE (type d'unité)
PRIMEDATADEVICE (type d'unité).

FRØMKEY (clé-1) et TØKEY (clé-2) permettent de limiter la recopie aux enregistrements dont la clé est comprise entre clé-1 et clé-2.

Avec FRØMADDRESS (adresse-1) et TØADDRESS (adresse-2) on fournit des adresses relatives au-lieu de clés.

FRØMNUMBER (Nombre-1) et TØNUMBER (Nombre-2) désignent le premier et le dernier des numéros d'enregistrements à recopier pour un fichier relatif VSAM.

SKIP (nombre-1) indique un nombre d'enregistrements à sauter avant de commencer la recopie, tandis que CØUNT (nombre-2) indique le nombre d'enregistrements qu'il faut recopier.

LA COMMANDE LISTCAT

La commande LISTCAT fournit une liste partielle ou exhaustive du contenu d'un catalogue VSAM. Cette commande est donc très utile pour contrôler les caractéristiques des fichiers inscrits au catalogue.

Le format général de cette commande est le suivant :

LISTCAT
[CATALØG (Nom de catalogue [/mot de passe] [DDName-1])]
[ØUTFILE (DDName-2)]
[ENTRIES (Nom-d'entrée [/mot de passe]...)]

```
[CLUSTER          ]
 DATA
 INDEX
 SPACE
 NØNVSAM
 ALTERNATEINDEX
 PATH
[UCAT             ]
```

```
[NAME             ]
 ALL
 VØLUME
[ALLØCATION       ]
```

CATALØG désigne le catalogue dont on désire lister le contenu.

ØUTFILE est à utiliser lorsqu'on désire que la liste soit éditée sur un fichier DDName-2 différent du fichier standard SYSPRINT.

ENTRIES permet de spécifier les entrées particulières du catalogue dont on désire la liste.

CLUSTER, DATA, INDEX, NØNVSAM, ALTERNATEINDEX, PATH et UCAT permettent de lister les informations relatives à un type particulier de fichier.

NAME signifie que seuls les noms des entrées du catalogue seront édités.

ALL signifie que toutes les informations seront éditées.

VØLUME, seules les informations concernant les volumes (numéros et types d'unités) seront éditées.

ALLØCATIØN liste les mêmes informations que VØLUME, plus les allocations physiques.

Exemple :

```
// LISTECAT JØB
// STEP01    EXEC PGM = IDCAMS
// STEPCAT   DD   DSN = MYCATAL, DISP = SHR
// SYSPRINT  DD   SYSØUT = A
// SYSIN     DD   *
        LISTCAT CATALØG (MYCATAL) —
                ALL;
/*
```

LA COMMANDE VERIFY

La commande VERIFY permet de vérifier l'état opérationnel d'un fichier VSAM. Dans le cas où le fichier n'aurait pas été fermé, à cause d'un incident d'exploitation lors du dernier traitement par exemple, il est impossible d'exploiter ce fichier à nouveau si on n'emploie pas au préalable la commande VERIFY qui restaure l'état du fichier.

Le format de cette commande est simplement.

VERIFY FILE (Nom DDName [/mot de passe]) ;

Le DDName correspond à la carte JCL de contrôle d'exploitation qui désigne le fichier.

Exemple :

```
// VERIFICH JØB
// JØBCAT   DD   DSN = MYCATAL, DISP = SHR
// STEP1    EXEC PGM = IDCAMS
// FIKSDS   DD   DSN = FIKSDS, DISP = ØLD
// SYSPRINT DD   SYSØUT = A
// SYSIN    DD   *
            VERIFY FILE (FIKSDS) ;
/*
```

LA COMMANDE PRINT

La commande PRINT permet d'éditer le contenu d'un fichier VSAM. Son format est pratiquement identique à celui de la commande REPRØ.

PRINT

INFILE (Nom DDName du fichier en entrée [/mot de passe])

[ENVIRØNMENT (DUMMY | liste de paramètres)]

ØUTFILE (Nom DDName du fichier en sortie [/mot de passe])

{ FRØMKEY (clé-1) | FRØMADDRESS (adresse-1) | FRØMNUMBER (nombre-1) | SKIP (nombre-1) }

{ TØKEY (clé-2) | TØADDRESS (adresse-2) | TØNUMBER (nombre-2) | CØUNT (nombre-2) }

[HEX | CHARACTER | DUMP] ;

HEX signifie que l'édition doit être en hexadécimal.

CHARACTER signifie que l'édition doit être en caractères.

DUMP signifie que l'édition doit être à la fois en hexadécimal et en caractères.

GESTION DES ERREURS — LA « STATUS KEY »

VSAM a apporté une modification essentielle dans la gestion des erreurs de traitements de fichiers. Qu'il s'agisse d'erreurs de logique dues au programmeur ou d'erreurs de transmissions sur le support mémoire externe, le traitement n'est plus automatiquement abandonné (ABEND) mais le contrôle est redonné au programme avec indication du type d'erreur dans un indicateur de format numérique externe à deux positions (PIC 99 USAGE DISPLAY).

Il devient donc indispensable, avec VSAM, d'associer une STATUS KEY à chaque fichier et de contrôler le contenu de cette clé après chaque exécution d'entrée-sortie ØPEN, CLØSE, READ, WRITE, REWRITE ou DELETE, faute de quoi d'énormes erreurs peuvent survenir sans que le programmeur ne s'en rende compte.

Invalid Key : Lorsque la clause INVALID KEY est stipulée, le traitement est dévié vers cette clause, mais il faut tout de même analyser la STATUS KEY pour connaître la raison exacte du déroutement.

Declarative EXCEPTION/ERROR : Le fonctionnement est identique à celui de la clause INVALID KEY et n'exclut pas l'analyse de la STATUS KEY. Notons en outre que ces déclaratives ne sont opérationnelles que lorsque l'ouverture ØPEN du fichier est correctement réalisée.

Valeurs prises par les STATUS KEY (en général).

0 0	Entrée-Sortie correcte.
0 2	» avec clé en double.
1 0	Fin de fichier ou fichier optionnel non présent à l'OPEN.
2 1	Invalid Key — Erreur de séquence.
2 2	» — Clé en double.
2 3	» — Enregistrement non trouvé.
2 4	» — Enregistrement hors des limites d'un fichier KSDS (Indexé).
3 0	I/0 Erreur — permanente.
3 4	» — Enregistrement hors des limites d'un fichier séquentiel ESDS.
9 0	Autre Erreur — Indéterminée.
9 1	» — Mot de passe.
9 2	» — Erreur logique (exemple : KSDS ouvert EXTEND et enregistrement ajouté de clé inférieure à la dernière des clés existantes ou réécriture d'un enregistrement ESDS de longueur différente de l'enregistrement initial).
9 3	» — Support non disponible.
9 4	» — Pas d'enregistrement en séquence.
9 5	» — Description de fichier invalide ou incomplète.
9 6	» — Pas de carte DD de contrôle d'exploitation pour le fichier.
9 7	» — Une commande VERIFY est nécessaire avant exploitation.

A l'ouverture ØPEN d'un fichier on peut en outre avoir les valeurs particulières suivantes :

9 0 — Un fichier KSDS indexé est ouvert ØUTPUT en accès DYNAMIC ou RANDØM.
— Un fichier ouvert I-Ø est créé mais non chargé.
— La génération dynamique d'un index secondaire par le programme COBOL (sans commande AMS explicite) n'a pas fonctionné.

9 2 — Le fichier à ouvrir est déjà ouvert.

9 3 — COBOL ne dispose pas de place mémoire centrale suffisante pour travailler.

9 5 — Le R. B. A. (Relative Byte Adress) d'un fichier RRDS en chargement ØUTPUT est différent de zéro.
— Les définitions de clés de fichier KSDS indexé du programme COBOL ne correspondent pas à celles enregistrées au catalogue VSAM.

9 6 — Il n'y a pas de carte JCL de contrôle d'exploitation pour une voie d'accès secondaire (PATH) qu'on veut ouvrir.

9 7 — Ouverture réussie pour un fichier KSDS Indexé avec index secondaire en mode I-Ø, EXTEND ou ØUTPUT.

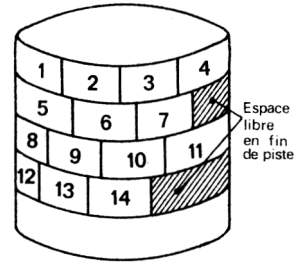
CHAPITRE V

L'ORGANISATION CONSÉCUTIVE VSAM - ESDS

PRINCIPES GÉNÉRAUX

Dans l'organisation consécutive ESDS, les données sont enregistrées dans l'ordre d'arrivée.

Normalement, il n'y a pas d'espace libre, sauf en fin de piste s'il n'est pas possible de loger un enregistrement (après l'enregistrement 7 figure ci-contre) ou en fin de fichier (enregistrement 14).



On accède aux enregistrements séquentiellement ou directement par adressage relatif compté en nombre d'octets, ce qui suppose qu'on connaisse cette adresse R. B. A. (Relative Byte Address) ou qu'on soit capable de la reconstituer.

Dans le cas d'enregistrements de longueur fixe, nous sommes ramenés pratiquement à l'organisation relative, à la différence près que l'adressage n'est pas un nombre d'enregistrements (traduit en CCHHR) mais directement un nombre d'octets.

En organisation ESDS, il est impossible de supprimer un enregistrement, mais il sera toujours possible de les oblitérer par un signe distinctif conventionnel (FF hexadécimal dans un octet par exemple).

On ne pourra pas plus intercaler des enregistrements (sauf si on a prévu des enregistrements préformatés vides) mais on pourra en ajouter en fin de fichier.

FORMAT GÉNÉRAL D'UN PROGRAMME ESDS

Le format général d'un programme en organisation consécutive ESDS est le suivant :

```
|A      |B
|IDEN   |TIFICATIØN DIVISIØN.
|       |  ⋮
|ENVI   |RØNMENT DIVISIØN.
|CØNF   |IGURATIØN SECTIØN.
|       |  ⋮
|INPU   |T-ØUTPUT SECTIØN.
|FILE   |-CØNTRØL.
|       |SELECT [ØPTIØNAL] Nom-de-fichier
|       |          ASSIGN TØ Nom-de-Fichier [Nom-de-Fichier]---
|       |                    Système-1      [  Système-2  ]
|       |      [ØRGANIZATIØN IS SEQUENTIAL]
|       |      [ACCESS MØDE IS SEQUENTIAL]
|       |      [PASSWØRD IS Zone Mot-de-Passe]
|       |      [FILE STATUS IS Nom-Donnée-1].
[ I-Ø-  |CØNTRØL.                                                ]
[       |[RERUN ØN Nom-de-fichier                                ]
[       |          de Sauvegarde                                 ]
[       |                EVERY n RECØRDS ØF Nom-de-Fichier]      ]
[       |[SAME [RECØRD]                                          ]
[       |          AREA FØR Nom-de-fichier-2 [Nom-de-fichier-3]----].]
|DATA   |DIVISION.
|FILE   |SECTIØN.
|FD     |Nom-de-fichier LABEL RECØRD IS STANDARD
|       |              DATA RECØRD IS Nom-Enregistrement-1
|       |                                [Nom-Enregistrement-2].
|01     |Nom-Enregistrement-1.
|       |  ⋮
|WØRK   |ING-STØRAGE SECTIØN.
[77     |Nom-Donnée-1 PIC 99.]
[77     |Mot-de-Passe PIC X(8).]
```

PRØCEDURE DIVISIØN.
DECLARATIVES.
Nom-de-Section SECTIØN. USE AFTER STANDARD { EXCEPTIØN / ERRØR } PRØCEDURE ØN { Nom-de-fichier / INPUT / ØUTPUT / I-Ø / EXTEND }.
ØPEN { INPUT / ØUTPUT / I-Ø / EXTEND } Nom-de-fichier.
[READ Nom-de-fichier RECØRD [INTØ Nom-Donnée] [AT END Ordre Impératif].]
[{ WRITE / REWRITE } Nom-d'Enregistrement [FRØM Nom-de-Zone].]
CLØSE Nom-de-fichier [WITH LØCK].

LES INSTRUCTIONS COBOL

SELECT [ØPTIØNAL] Nom-de-fichier
ASSIGN TØ Nom-de-fichier Système 1 [Nom-de-fichier Système 2]

Cette clause établit une correspondance entre le Nom-de-fichier désigné en Data Division et le nom de fichier désigné dans les cartes de contrôle JCL du système d'exploitation.

Le Nom de fichier système s'écrit sous la forme :

AS – Nom de fichier Symbolique de JCL

Exemple : SELECT FIESDS ASSIGN TØ : AS – FIESDS.

En système DOS/VS, le Nom-de-fichier Système s'écrit sous la forme:

SYSnnn – Numéro de type d'unité – AS

Exemple : SELECT FIESDS ASSIGN TØ : SYS015-2314-AS.

SYSnnn correspond au Nom-Symbolique de fichiers des cartes de contrôle JCL.

Numéro de type d'unité : 2314, 3330, 3340, etc.

En fait, les numéros de type d'unité ne sont pas pris en compte par COBOL qui utilise les informations inscrites au catalogue.

Seul AS est nécessaire pour désigner le fichier VSAM séquentiel.

[ØRGANIZATIØN IS SEQUENTIAL] est l'option par défaut. Cette instruction n'est donc pas indispensable.

[ACCESS MØDE IS SEQUENTIAL] Cette instruction n'est pas indispensable non plus puisque c'est l'option par défaut et qu'on ne peut aujourd'hui accéder à un fichier ESDS que séquentiellement.

[PASSWØRD IS Mot-de-Passe] Cette instruction est associée à la déclaration d'une zone alphanumérique de 8 octets en WØRKING-STØRAGE SECTIØN.

Exemple : PASSWØRD IS MØTPASS.
et 77 MØTPASS PIC X(8).

Lorsque cette option est programmée, il faut que la zone Mot-de-Passe associée contienne le bon mot de passe donnant accès au fichier ESDS au moment de l'exécution du programme.

On pourra écrire par exemple :

```
DISPLAY 'DØNNEZ MØT DE PASSE XXXXXXXX'
ACCEPT MØTPASS.
```

[FILE STATUS IS Nom-Donnée-1] Cette instruction désigne la STATUS KEY, zone donnée de deux caractères, qui indique les conditions de réalisation de toute opération d'entrée-sortie sur le fichier correspondant. La STATUS KEY doit être décrite en WØRKING-STØRAGE, sous forme numérique externe.

Exemple : FILE STATUS IS STATCLE.
et 77 STATCLE PIC 99.

Voir le paragraphe GESTION DES ERREURS. LA « STATUS KEY » au chapitre IV pour plus de détails.

[RERUN ØN ~~~] Comme nous l'avons déjà vu, cette clause sert à provoquer des points de reprise tous les *n* enregistrements, c'est-à-dire recopier l'ensemble de la mémoire centrale sur le fichier de sauvegarde pour permettre une reprise éventuelle après incident. Le Nom-de-fichier de sauvegarde doit être bâti comme les noms de fichier système de la clause SELECT.

[SAME [RECØRD] AREA ~~~] Cette clause permet d'économiser de la mémoire centrale en réservant une seule zone buffer d'entrée-sortie pour plusieurs fichiers, à condition que ces fichiers ne soient pas utilisés simultanément.

FILE SECTIØN. La description de fichier FD est obligatoire, mais seules les clauses DATA RECØRD et LABEL RECØRD sont exigées car toutes les autres informations doivent exister dans le catalogue VSAM.

La clause LABEL RECØRD n'est d'ailleurs exigée que pour la syntaxe.
ØPEN~~~ Tout fichier ESDS doit être ouvert en :

INPUT avant le premier READ de lecture
ØUTPUT avant le premier WRITE d'écriture

I–Ø avant les verbes READ de lecture puis REWRITE d'écriture.
EXTEND avant le premier verbe d'écriture WRITE. Ce verbe EXTEND signifie qu'on désire ajouter des enregistrements en fin de fichier séquentiellement.

A condition qu'il y ait des ordres CLØSE de fermeture entre, il sera possible d'ouvrir successivement un fichier dans chacun des modes INPUT, ØUTPUT, I-Ø ou EXTEND.

READ Nom-de-fichier RECØRD
[INTØ Nom-Donnée] [AT END Ordre Impératif]

L'instruction de lecture est conditionnée par le positionnement d'un pointeur d'enregistrement en cours. Ce positionnement est déterminé par un ØPEN ou READ précédent. Si, par exemple, la lecture précédente s'est mal déroulée, le pointeur ne contient pas d'information valide, ce qui rend la lecture impossible (STATUS KEY = 94).

En fin de fichier, on a mise-à-jour de la STATUS KEY, puis exécution de l'ordre impératif si l'instruction AT END est programmée ; ou exécution dans le cas contraire d'une section DECLARATIVES-USE AFTER ERRØR. Ceci explique le fait que la clause AT END ne soit pas indispensable.

{ WRITE / REWRITE } Nom d'enregistrement [FRØM Nom-de-zone].

WRITE est utilisé pour une création de fichier (option ØUTPUT) ou une extension (EXTEND).

REWRITE est utilisé après une ordre de lecture READ pour réécrire un enregistrement modifié. Rappelons que la réécriture ne sera possible que si la longueur de l'enregistrement est maintenue constante.

CLØSE Nom-de-fichier [WITH LØCK] Tout fichier doit être fermé avant la fin de travail pour permettre au système d'exploitation d'effectuer les contrôles nécessaires.

WITH LØCK est une précaution particulière pour éviter que le même fichier soit à nouveau ouvert dans le même programme.

EXEMPLE DE PROGRAMME EN ORGANISATION VSAM CONSÉCUTIVE

L'accès séquentiel étant seul autorisé en organisation VSAM consécutive, nous nous bornerons à donner un exemple de création de fichier.

Nous supposons lire des cartes dont nous vérifions le code carte 'CG1' et que nous écrivons en organisation VSAM consécutive sur disque.

La valeur 00 de la STATUS KEY permet de contrôler l'écriture des enregistrements.

```
//STEP00   EXEC PGM=IDCAMS
//STEPCAT  DD   DSN=MASCAT,UNIT=3330,VOL=SER=SHR002,DISP=SHR
//SYSPRINT DD   SYSOUT=A
//MYSPACE  DD   UNIT=SYSDA,VOL=SER=SHR001,DISP=SHR
//SYSIN    DD   *

 DEFINE  SPACE                          -
         (FILE(MYSPACE)                 -
          VOL(SHR001)                   -
          CYL(2,2));

 DEFINE  CLUSTER                        -
         (NAME(FIESVSAM)                -
          NONINDEXED                    -
          OWNER(BONNIN)                 -
          VOL(SHR001)                   -
          CYL(1,1)                      -
          RECORDSIZE(80 80))            -
         DATA                           -
         (NAME(FIESVSAM.DATA));
```

```
001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID. CREAESDS.
001040 ENVIRONMENT  DIVISION.
001050 INPUT-OUTPUT SECTION.
001060 FILE-CONTROL.
001070     SELECT  FICARTE     ASSIGN TO  UT-S-FICARTE.
001080     SELECT  FIESVSAM    ASSIGN TO     FIESVSAM
001110                         FILE STATUS STATUSIO.
001120 DATA  DIVISION.
001130 FILE SECTION.
001140 FD  FICARTE     RECORDING  MODE F
001150                LABEL RECORD OMITTED
001160                DATA RECORD CARTE.
001170 01  CARTE.
001180     02  COCART  PIC     XXX.
001190     02  FILLER  PIC     X(77).
001200 FD  FIESVSAM    LABEL RECORD STANDARD
001210                DATA RECORD DISQUE.
001220 01  DISQUE.
001230     02  FILLER  PIC     X(80).
001300 WORKING-STORAGE  SECTION.
001310 77  STATUSIO    PIC     99.
002010 PROCEDURE  DIVISION.
002020 DEBUT.
002030     OPEN  INPUT  FICARTE  OUTPUT  FIESVSAM.
002032     IF STATUSIO NOT = 00
002034     DISPLAY "OPEN IMPOSSIBLE  STATUS= " STATUSIO
002036     CLOSE FICARTE FIESVSAM STOP RUN.
002040 LECTURE.
002050     READ FICARTE AT END GO TO FINTRA.
002060     IF COCART NOT = "CG1" GO TO ERREUR1.
002070     WRITE  DISQUE  FROM CARTE.
002080     IF  STATUSIO NOT = 00  GO TO ERREUR2.
002090     GO TO LECTURE.
002100 ERREUR1.
002110     DISPLAY "CARTE INVALIDE"  DISPLAY  CARTE
002120     GO TO LECTURE.
002130 ERREUR2.
002140     DISPLAY "ECRITURE IMPOSSIBLE" DISPLAY CARTE
002150     GO TO LECTURE.
002160 FINTRA.
002170     CLOSE  FICARTE  FIESVSAM.
002180     DISPLAY  "FIN DE TRAVAIL"
002190     STOP  RUN.
```

CHAPITRE VI

L'ORGANISATION INDEXÉE VSAM - KSDS

PRINCIPES GÉNÉRAUX

Nous avons vu qu'un fichier en organisation indexée VSAM est composé, en fait, de deux fichiers :

. le fichier des données
. le fichier index

regroupés sous le nom de CLUSTER

Le fichier des données :

Il comporte des enregistrements de longueur fixe ou variable et des espaces libres pour ajouts éventuels (voir structure logique des fichiers VSAM). Les enregistrements logiques sont rangés dans l'ordre croissant d'une clé appelée RECØRD KEY et sont aussi repérés par l'adresse relative R. B. A., comme les fichiers ESDS. Mais rappelons que, à ce jour, le R. B. A. n'est pas accessible en COBOL.

Le fichier Index :

Il contient les clés des enregistrements et des pointeurs associés permettant l'accès direct. Il est organisé en Control Interval et Control Area comme tous les fichiers VSAM.

Utilisation du fichier Index :

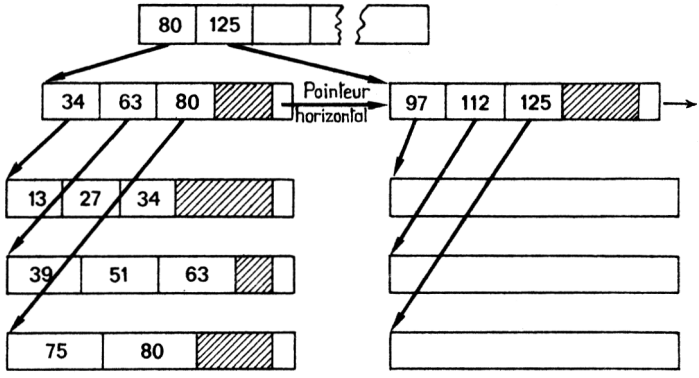
Le principe de recherche d'un enregistrement est sensiblement identique à celui de l'organisation SEQUENTIEL INDEXE, mais la NØMINAL

KEY n'existant pas en VSAM, c'est à partir de la RECØRD KEY que sont faites les comparaisons de recherches.

Le fichier index est constitué d'une succession de tables, de plus en plus détaillées au fur et à mesure qu'on approche de l'égalité avec les clés des enregistrements du fichier données.

La table de plus haut niveau pointe sur une table de niveau inférieur dont les éléments pointent à leur tour chacun sur une table de niveau inférieur, jusqu'au niveau le plus bas de l'index qui pointe finalement sur un Control Interval (C. I.) du fichier des données qui est alors exploré en séquence pour rechercher le bon enregistrement.

Par exemple (figure ci-dessous), pour rechercher l'enregistrement de clé 51, on passe par les cases index 80 → 34 et 63 puis 63 → 39 et 51.



Outre les pointeurs « verticaux » dont il est question ci-dessus, il existe aussi un système de pointeurs horizontaux qui permettent, lors de traitements séquentiels de sauter des groupes d'enregistrements.

Compression de clés :

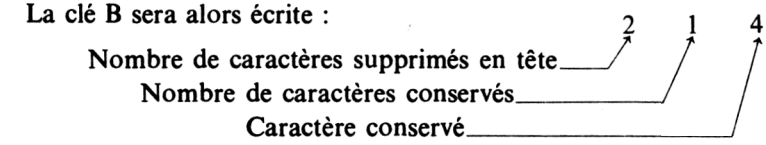
Les clés stockées dans le fichier index sont comprimées afin d'activer le processus de recherche.

Par exemple, pour des clés consécutives

A =	22	3	45
B =	~~22~~	4	~~30~~
C =	~~22~~	5	~~13~~

On supprime les deux premiers caractères de la clé B puisque identiques à ceux de la clé A, ainsi que les deux derniers puisque, pour la recherche par comparaisons, seul le troisième caractère est significatif.

La clé B sera alors écrite :



Nous n'insisterons pas plus sur ce processus de compression qui n'influe en rien sur la programmation COBOL.

FORMAT GÉNÉRAL D'UN PROGRAMME EN ORGANISATION INDEXÉE KSDS

```
A       B
 IDENTIFICATIØN DIVISIØN.
 ENVIRØNMENT DIVISIØN.
 CØNFIGURATIØN SECTIØN.
 INPUT-ØUTPUT SECTIØN.
  FILE-CØNTRØL.
        SELECT Nom-de-fichier ASSIGN TØ Nom-de-fichier Système-1 [Nom-de-fichier Système-2]
                        ØRGANIZATIØN IS INDEXED
                        [ACCESS MØDE IS {SEQUENTIAL | RANDØM | DYNAMIC}]
                         RECØRD KEY IS Nom-donnée-2
                        [PASSWØRD IS Zone Mot-de-passe]
                        [FILE STATUS IS Nom-Donnée-1].
[   I-Ø-CØNTRØL.                                                          ]
[       [RERUN ØN Nom-de-fichier de sauvegarde
                        EVERY n RECØRDS ØF Nom-de-Fichier]                ]
[       [SAME [RECØRD] AREA FØR Nom-de-Fichier-2
                        [Nom-de-Fichier-3] ---]..                         ]
  DATA  DIVISIØN.
   FILE SECTIØN.
  FD    Nom-de-Fichier LABEL RECØRD IS STANDARD
                [DATA RECØRD IS Nom-Enregistrement-1
                        [Nom-Enregistrement-2].]
  01    Nom-Enregistrement-1.
 WØRKING-STØRAGE SECTIØN.
[ 77    Nom-Donnée-1 PIC 99.]
[ 77    Zone Mot-de-Passe PIC X(8).]
```

```
A       B
PRØCEDURE DIVISIØN.
DECLARATIVES.
Nom-de-Section SECTIØN. USE AFTER STANDARD { EXCEPTIØN | ERRØR }
                                  PRØCEDURE ØN { Nom-de-Fichier | INPUT | ØUTPUT | I-Ø | EXTEND } .
ØPEN { INPUT | ØUTPUT | I-Ø | EXTEND } Nom-de-Fichier.
[START Nom-de-Fichier
 KEY IS { EQUAL TØ | GREATER THAN | > | NØT LESS THAN | NØT < } Nom-Donnée [INVALID KEY Ordre Impératif]]
[READ Nom-de-Fichier [NEXT] RECØRD [INTØ Nom-Donnée-3]
                                  [AT END Ordre Impératif | INVALID KEY Ordre Impératif] .]
[{ WRITE | REWRITE } Nom-d'Enregistrement
      [FRØM Nom-de-Zone] [INVALID KEY Ordre-Impératif].]
[DELETE Nom-de-Fichier RECØRD
                                  [INVALID KEY Ordre-Impératif].]
CLØSE Nom-de-Fichier [WITH LØCK].
```

LES INSTRUCTIONS COBOL

SELECT Nom-de-fichier ASSIGN TØ Nom-de-fichier Système 1 [Nom-de-fichier Système 2]

Cette clause établit une correspondance entre le Nom-de-fichier désigné en Data Division et le Nom-de-fichier désigné dans les cartes de contrôle JCL du système d'exploitation.

Le Nom-de-fichier Système s'écrit simplement

[Commentaire]- Nom-de-fichier Symbolique de JCL

Exemple :

SELECT FIKSVSAM ASSIGN TØ FIKSVSAM.

On remarque que la partie organisation (I de séquentiel indexé par exemple) est absente.

En système DOS/VS le Nom-de-fichier Système s'écrit sous la forme

$$\text{SYSnnn} - \text{DA} - \begin{matrix}\text{Numéro de} \\ \text{type d'unité}\end{matrix}$$

Exemple : SELECT FIKSVSAM ASSIGN TØ SYS015-DA-3330.

SYSnnn correspond au Nom-Symbolique de fichier des cartes de contrôle JCL.

Numéro de type d'unité : 2314, 3330, 3340, etc.

En fait, DA et le numéro d'unité ne sont donnés que pour la syntaxe par COBOL utilise les informations inscrites au catalogue.

ØRGANIZATIØN IS INDEXED est indispensable pour désigner un fichier en organisation indexée.

[ACCESS MØDE IS { SEQUENTIAL / RANDØM / DYNAMIC }] Cette clause indique le type d'accès au fichier utilisé dans le programme.

SEQUENTIAL, option par défaut, signifie que le fichier sera traité séquentiellement suivant l'ordre normal des clés croissantes.

RANDØM signifie que le fichier sera traité en accès direct par recherche de clé.

DYNAMIC signifie que le fichier sera traité à la fois séquentiellement et en accès direct.

RECØRD KEY IS Nom-Donnée-2 Cette clause désigne la clé dans l'enregistrement du fichier. Nom-Donnée-2 doit donc appartenir à la description de l'enregistrement, être alphanumérique et correspondre à la clé définie lors de la création du fichier avec le langage A. M. S.

[PASSWØRD IS Zone Mot-de-Passe] Voir l'organisation consécutive ESDS.

[FILE STATUS IS Nom-Donnée-1] Cette instruction est identique à celle de l'organisation consécutive ESDS, mais la STATUS KEY peut prendre des valeurs supplémentaires en organisation indexée.

Voir le paragraphe GESTION DES ERREURS. LA « STATUS KEY » au chapitre IV pour plus de détails.

[RERUN ØN~~~]
[SAME [RECØRD] AREA~~~]
FILE SECTIØN.
ØPEN~~~~~~~

Voir l'organisation consécutive ESDS

[START Nom-de-Fichier~~~] Cette instruction ne peut être utilisée qu'en traitement séquentiel de lecture INPUT ou mise-à-jour I-Ø.

On doit avoir déclaré ACCESS MØDE SEQUENTIAL ou DYNAMIC et avoir ouvert le fichier en INPUT ou I-Ø.

Cette instruction est utilisée lorsqu'on ne désire pas commencer le traitement au début du fichier mais à partir d'une valeur de clé désignée par égalité, supérieur-à ou non-inférieur-à.

Il est possible, comme en organisation séquentielle indexée, de désigner une clé de groupe, c'est-à-dire une clé tronquée à droite.

Par exemple :

```
START FIKSVSAM KEY NØT < CLEGRØUP.
avec 77 CLEGRØUP PIC XX VALUE "22".
```

Ceci permet de commencer le traitement avec les clés 22XXXX.

[READ Nom-de-fichier [NEXT] RECØRD
[INTØ Nom-Donnée] [AT END Ordre Impératif].

Ce format est à utiliser en accès séquentiel (ACCESS MØDE SEQUENTIAL). On obtient alors l'enregistrement suivant en séquence.

L'option NEXT *ne doit être* utilisée qu'avec ACCESS MØDE DYNAMIC pour spécifier un accès en séquence et non pas au hasard, puisque DYNAMIC permet les deux formes d'accès.

Avant tout ordre de lecture READ, il faut avoir positionné le *pointeur d'enregistrement en cours* par un ordre ØPEN, START ou READ précédent. Si le pointeur n'est pas correctement positionné (suite à une précédente erreur de lecture séquentielle), le pointeur ne contient pas d'information valide, ce qui rend la lecture impossible (STATUS KEY = 94).

En fin de fichier, on a mise-à-jour de la STATUS KEY puis exécution de l'ordre impératif lorsque l'instruction AT END est programmée ; sinon il y a exécution d'une section déclaratives USE AFTER ERRØR.

On pourra donc, le cas échéant, remplacer la clause AT END par une section Declaratives.

[READ Nom-de-fichier RECØRD [INTØ Nom-Donnée-3]
[INVALID KEY Ordre Impératif]].

Ce format est à utiliser avec ACCESS MØDE RANDØM ou ACCESS MØDE DYNAMIC, c'est-à-dire en accès direct.

Avant exécution de ce READ, il faut avoir chargé la RECØRD KEY de la valeur de la clé à rechercher, car c'est à partir de cette RECØRD KEY qu'est effectuée la recherche dans l'index VSAM.

Notons que, lorsque l'enregistrement est trouvé, le pointeur d'enregistrement en cours est positionné pour rendre cet enregistrement disponible.

Si l'enregistrement ne peut être trouvé, alors la condition INVALID KEY est activée ainsi que la STATUS KEY, ce qui permet de traiter très complètement cette anomalie.

On pourra aussi omettre la clause INVALID KEY, à condition d'avoir programmé une section Declaratives

$$\underline{\text{USE}}\ \underline{\text{AFTER}}\ \text{STANDARD}\ \left\{\begin{array}{l}\underline{\text{EXCEPTIØN}}\\ \underline{\text{ERRØR}}\end{array}\right\}$$

En fait, le programmeur a le choix :

. Programmer autant de séquences INVALID KEY qu'il y a de cas.

. Programmer une section Declaratives générale analysant toutes les valeurs possibles de STATUS KEY.

[WRITE Nom d'enregistrement
[FRØM Nom-de-zone] [INVALID KEY Ordre Impératif].]

Cette clause exige un fichier ouvert en mode ØUTPUT, I-Ø ou EXTEND.

La STATUS KEY est positionnée à chaque exécution de l'ordre WRITE.

C'est à partir de la valeur contenue dans la RECØRD KEY que l'enregistrement sera correctement écrit dans le fichier.

En mode séquentiel, les clés devront être classées dans l'ordre croissant et uniques.

En mode direct, les clés devront être uniques.

Comme pour l'ordre READ, on pourra contrôler la bonne exécution de l'ordre WRITE par la clause INVALID KEY ou par une section Declaratives.

[REWRITE Nom-d'Enregistrement
[FRØM Nom-de-Zone] [INVALID KEY Ordre Impératif].]

Cette clause doit être utilisée après un ordre READ pour réécrire un enregistrement modifié (on peut faire varier la longueur de l'enregistrement).

Cette clause exige un fichier ouvert en mode I-Ø.

On pourra ici aussi contrôler la bonne exécution de l'ordre REWRITE par la clause INVALID KEY ou par une section DECLARATIVES.

[DELETE Nom-de-fichier RECØRD [INVALID KEY Ordre Impératif]].

Cette clause exige un fichier ouvert en mode I-Ø, avec accès séquentiel ou direct. Elle permet de supprimer logiquement un enregistrement d'un fichier indexé VSAM.

En accès séquentiel, il faut au préalable avoir effectué une lecture READ de l'enregistrement à supprimer (la clause INVALID KEY n'a pas de sens ici).

En accès direct, la RECØRD KEY doit avoir été chargée (la clause INVALID KEY a une signification).

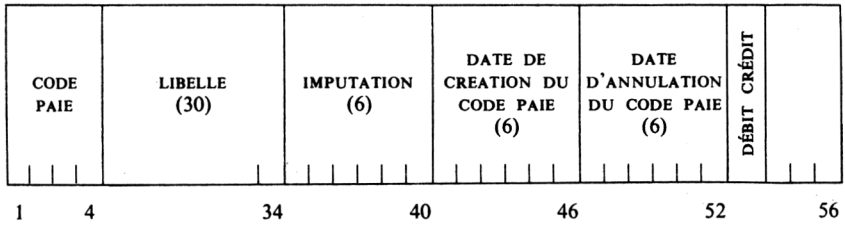
CLØSE Nom-de-Fichier [WITH LØCK]

Tout fichier doit être fermé avant la fin de travail pour permettre au système d'exploitation d'effectuer les contrôles nécessaires.

WITH LØCK est une précaution particulière pour éviter que le même fichier soit à nouveau ouvert dans le même programme.

EXEMPLES DE PROGRAMMATION VSAM

Pour illustrer l'organisation VSAM Indexée, nous allons reprendre l'exemple du fichier des rubriques de paie du chapitre séquentiel Indexé (mais sans position HIGH-VALUE).



Pour commencer, nous devons définir grâce au langage AMS :

. Un catalogue utilisateur.
. Un espace VSAM.
. Un CLUSTER Index = FIKSVSAM.

```
//STEP00  EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=A
//DDCAT    DD  UNIT=2314,VOL=SER=210VS1,DISP=SHR
//DDUSE    DD  UNIT=2314,VOL=SER=436000,DISP=SHR
//SYSIN    DD  *

 DEFINE  USERCATALOG          -
         (NAME(USERCATA)      -
          FILE(DDUSE)         -
          VOL(436000)         -
          CYL(2,2)            -
          WRITECHECK)         -
         CATALOG(MASTCTLG/MASTER)}
```

```
//STEP01  EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=A
//STEPCAT  DD  DSN=USERCATA,VOL=SER=436000,UNIT=2314,DISP=SHR
//DDSPC    DD  UNIT=2314,VOL=SER=437000,DISP=SHR
//SYSIN    DD  *

 DEFINE  SPACE                -
         (FILE(DDSPC)         -
          VOL(437000)         -
          CYL(2,2))
```

```
//STEP02  EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=A
//STEPCAT  DD  DSN=USERCATA,VOL=SER=436000,UNIT=2314,DISP=SHR
//SYSIN    DD  *

 DEFINE  CLUSTER              -
         (NAME(FIKSVSAM)      -
          INDEXED KEYS(4 0)   -
          VOL(437000)         -
          CYL(1,1)            -
          FSPC(20,10)         -
          RECORDSIZE(56 56)   -
          WRITECHECK)}
```

En COBOL maintenant, nous allons charger les enregistrements à partir de cartes ayant P21 pour codes cartes. Le traitement est séquentiel.

```
//STEP03 EXEC  COBUCLG,
// PARM.COB='NODECK,PMAP,VERB,XREF,QUOTE'
//SYSLIB DD DISP=SHR,DSN=SMABTP.MACLIB,VOL=SER=SMAD05,UNIT=3330
//COB.SYSIN  DD  *

001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID. CREKSVS.
001050 ENVIRONMENT  DIVISION.
001060 INPUT-OUTPUT  SECTION.
001070 FILE-CONTROL.
001080     SELECT  FICARTE     ASSIGN TO   UT-S-FICARTE.
001090     SELECT  FIKSVSAM    ASSIGN TO   DA-FIKSVSAM
001095                         ORGANIZATION INDEXED
001100                         ACCESS IS SEQUENTIAL
001105                         FILE STATUS STATKEY
001110                         RECORD KEY IS CODPAIE.
001120 DATA  DIVISION.
001130 FILE  SECTION.
001140 FD  FICARTE      RECORDING MODE  F
001150                 LABEL RECORD OMITTED
001160                 DATA RECORD CARTE.
001170 01  CARTE.
001180     02  COCART  PIC      XXX.
001190     02  CODEP   PIC      9(4).
001200     02  ENREG   PIC      X(49).
001210     02  FILLER  PIC      X(24).
001240 FD  FIKSVSAM     LABEL RECORD STANDARD
001250                 DATA RECORD DISQUE.
001260 01  DISQUE.
002020     02  CODPAIE PIC      9(4).
002030     02  RESTE   PIC      X(52).
002032 WORKING-STORAGE SECTION.
002034 01  STATKEY  PIC  99.
002040 PROCEDURE  DIVISION.
002050 DEBUT.
002060     OPEN  INPUT FICARTE   OUTPUT FIKSVSAM.
          IF STATKEY NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= "
                      STATKEY  GO TO FINTRA.
002070 LECTURE.
002080     READ FICARTE AT END GO TO FINTRA.
002090     IF COCART  NOT = "P21"  GO TO ER1.
002100     MOVE CODEP TO CODPAIE
002110     MOVE ENREG TO RESTE
002120     WRITE DISQUE  INVALID KEY GO TO ER2.
          IF STATKEY NOT = 00 DISPLAY "WRITE IMPOSSIBLE CODEP = "
                              CODEP  "STATUS= " STATKEY GO TO FINTRA.
002130     GO TO LECTURE.
002140 ER1.
002150     DISPLAY "CARTE INVALIDE"  DISPLAY CARTE  GO TO LECTURE.
002160 ER2.
002170     DISPLAY "ERREUR SEQUENCE"   DISPLAY CARTE   GO TO LECTURE.
002180 FINTRA.
002190     CLOSE  FICARTE  FIKSVSAM
002200     DISPLAY "FIN DE TRAVAIL"    STOP RUN.

//GO.FIKSVSAM  DD  DSN=GGDV.RES.FIKSVSAM,DISP=NEW
//GO.SYSOUT    DD  SYSOUT=A
//GO.SYSUDUMP  DD  SYSOUT=A
//GO.FICARTE   DD  *
```

En COBOL encore, nous allons éditer le contenu du fichier en Organisation VSAM indexée FIKSVSAM.

```
//STEP04 EXEC  COBUCLG,PARM.COB='NODECK'
//COB.SYSIN  DD  *

001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID.  EDITKSVS.
001050 ENVIRONMENT  DIVISION.
001060 INPUT-OUTPUT  SECTION.
001070 FILE-CONTROL.
001080     SELECT  FIKSVSAM ASSIGN TO    FIKSVSAM
001085             ORGANIZATION INDEXED
001090             ACCESS IS SEQUENTIAL
001095                          FILE STATUS STATKEY
001100             RECORD KEY IS CODEPAIE.
001110     SELECT  IMPRI ASSIGN TO UT-S-IMPRI.
001120 DATA  DIVISION.
001130 FILE  SECTION.
001160 FD  FIKSVSAM  LABEL RECORD STANDARD
001170               DATA RECORD DISQUE.
001180 01  DISQUE.
001200     02  CODEPAIE PIC  9(4).
001210     02  RESTE    PIC  X(52).
001220 FD  IMPRI  RECORDING MODE F
001230            LABEL RECORD OMITTED
001240            DATA RECORD LIGNE.
001250 01  LIGNE.
001260     02  SAUT    PIC  X.
001190     02  LIDET   PIC  X(132).
001200 WORKING-STORAGE SECTION.
001210 01  STATKEY  PIC  99.
002010 PROCEDURE  DIVISION.
002020 DEBUT.
002030     OPEN INPUT FIKSVSAM  OUTPUT IMPRI.
002032     IF STATKEY NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= "
002034                    STATKEY  GO TO FINTRA.
002040 LECTURE.
002050     READ FIKSVSAM AT END GO TO FINTRA.
002052     IF STATKEY NOT = 00 DISPLAY "LECTURE IMPOSSIBLE STATUS=
002054                    STATKEY  GO TO FINTRA.
002060     MOVE DISQUE TO LIDET
002070     WRITE LIGNE AFTER  1.
002080     GO TO LECTURE.
002090 FINTRA.
002100     CLOSE FIKSVSAM  IMPRI
002110     DISPLAY  "FIN  DE  TRAVAIL"
002120     STOP RUN.


//GO.FIKSVSAM  DD  DSN=FIKSVSAM,DISP=OLD
//GO.SYSOUT    DD  SYSOUT=A
//GO.SYSUDUMP  DD  SYSOUT=A
//GO.IMPRI     DD  SYSOUT=A,
//   DCB=(RECFM=FBA,LRECL=133,BLKSIZE=133)
```

En COBOL enfin, nous allons effectuer une mise à jour des enregistrements en accès direct RANDOM.

Le principe général de traitement est semblable à celui du traitement séquentiel indexé.

A partir de l'indicatif de l'enregistrement CARTE de mise à jour, on charge la RECØRD KEY pour lire le fichier permanent en organisation VSAM indexée.

- Si l'enregistrement du fichier permanent est trouvé, on peut réaliser une modification par REWRITE, ou une suppression par DELETE.
- Si l'enregistrement n'est pas trouvé, le mouvement doit être une création par WRITE.

On remarquera particulièrement qu'on utilise le verbe DELETE dans le cas de suppressions.

```
 /STEP05 EXEC  COBUCLG,PARM.COB='NODECK'
//COB.SYSIN  DD  *


001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID.  RANDKSVS.
001050 ENVIRONMENT  DIVISION.
001055 INPUT-OUTPUT  SECTION.
001060 FILE-CONTROL.
001070     SELECT  FIKSVSAM     ASSIGN TO      FIKSVSAM
001075                          ORGANIZATION INDEXED
001080                          ACCESS IS RANDOM
001085                          FILE STATUS STATKEY
001090                          RECORD KEY IS CODPAIE.
001120     SELECT  FICARTE      ASSIGN TO   UT-S-FICARTE.
001130     SELECT  IMPRI        ASSIGN TO   UT-S-IMPRI.
001160 DATA  DIVISION.
001170 FILE  SECTION.
001200 FD  FIKSVSAM     LABEL  RECORD STANDARD
001210                 DATA RECORD DISQUE.
001220 01  DISQUE.
001240     02  CORPS.
001250         04  CODPAIE PIC     9(4).
001260         04  LIBEL   PIC     X(30).
001270         04  IMPUT   PIC     X(6).
002010         04  DATCRE  PIC     X(6).
002020         04  DATFIN  PIC     X(6).
002030         04  DEBCRE  PIC     X.
002040         04  FILLER  PIC     XXX.
```

```
002050 FD  FICARTE RECORDING  F
002060             LABEL RECORD OMITTED
002070             DATA RECORD CARTE.
002080 01  CARTE.
002090     02  COCART  PIC     XXX.
002100     02  CACORP.
002110         04  CODEP   PIC     9(4).
002120         04  CARLIB  PIC     X(30).
002130         04  CARIMP  PIC     X(6).
002140         04  CARCRE  PIC     X(6).
002150         04  CARFIN  PIC     X(6).
002160         04  CARDB   PIC     X.
002170         04  FILLER  PIC     X(23).
002180         04  CODMAJ  PIC     X.
002190 FD  IMPRI   RECORDING MODE F
002200             LABEL RECORD OMITTED
002210             DATA RECORD LIGNE.
002220 01  LIGNE   PIC   X(133).
002230 WORKING-STORAGE  SECTION.
002240 01  STATKEY  PIC  99.
003010 01  LI1.
003020     02  TITRE   PIC    X(17).
003030     02  ENR1    PIC    X(56).
003040     02  FILLER  PIC    X(60)  VALUE SPACE.
003050 01  LI2.
003060     02  FILLER  PIC X(23) VALUE "CARTE MISE A JOUR = ".
003070     02  ENR2    PIC X(80).
003080     02  FILLER  PIC X(30)  VALUE  SPACE.
003090 PROCEDURE  DIVISION.
003100 DEBUT.
003110     OPEN  INPUT     FICARTE  OUTPUT IMPRI  I-O  FIKSVSAM.
003112     IF STATKEY NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= "
003114                    STATKEY  GO TO FINTRA.
003120 LECTURE.
003130     READ FICARTE AT END CLOSE FIKSVSAM FICARTE IMPRI  STOP RUN.
003140     MOVE CODEP TO CODPAIE
003150     MOVE CARTE TO ENR2  WRITE LIGNE FROM LI2
003160     READ FIKSVSAM  INVALID KEY  GO TO INCONNU.
003170     MOVE DISQUE TO ENR1     MOVE "AVANT" TO TITRE
003180     WRITE LIGNE FROM LI1
003200     IF CODMAJ = "C"   GO TO ER1.
003210     IF CODMAJ = "M"   GO TO MODIFICATION.
003220     IF CODMAJ = "S"   GO TO SUPPRESSION.
003230     MOVE "ERREUR2" TO TITRE
003240     MOVE "CODE MAJ INCONNU" TO ENR1
003250     WRITE LIGNE FROM LI1    GO TO LECTURE.
004010 ER1.
004020     MOVE "ERREUR1" TO TITRE
004030     MOVE "MAJ A TORT"  TO  ENR1.
004040     WRITE LIGNE FROM LI1    GO TO LECTURE.
004050 INCONNU.
004060     IF  CODMAJ NOT = "C"  GO TO ER1.
```

```
004070 CREATION.
004090     MOVE  CACORP TO CORPS
004100     WRITE DISQUE  INVALID KEY GO TO ER3.
004110     GO TO ECRIT.
004120 SUPPRESSION.
004140     DELETE FIKSVSAM INVALID KEY GO TO ER3.
004150     MOVE "APRES" TO TITRE   MOVE DISQUE TO ENR1.
004160 ECRIT.
004170     WRITE LIGNE FROM LI1    GO TO LECTURE.
004180 ER3.
004190     MOVE  "ERREUR3"  TO TITRE
004200     MOVE  "ECRITURE IMPOSSIBLE"  TO ENR1
004210     GO TO ECRIT.
004220 MODIFICATION.
004230     IF  CARLIB  NOT = SPACE MOVE    CARLIB  TO  LIBEL.
004240     IF  CARIMP  NOT = SPACE MOVE    CARIMP  TO  IMPUT.
004250     IF  CARCRE  NOT = SPACE MOVE    CARCRE  TO  DATCRE.
004260     IF  CARFIN  NOT = SPACE MOVE    CARFIN  TO  DATFIN.
004270     IF  CARDB   NOT = SPACE MOVE    CARDB   TO  DEBCRE.
005010     REWRITE  DISQUE  INVALID KEY GO TO ER3.
005020     MOVE  "APRES" TO TITRE  MOVE DISQUE TO ENR1
005030     GO TO ECRIT.


//GO.FIKSVSAM  DD  DSN=FIKSVSAM,DISP=OLD
//GO.SYSUDUMP  DD  SYSOUT=A
//GO.SYSOUT    DD  SYSOUT=A
//GO.IMPRI     DD  SYSOUT=A
//GO.FICARTE   DD  *
```

CHAPITRE VII

L'ORGANISATION RELATIVE RRDS (1)

PRINCIPES GÉNÉRAUX

L'organisation Relative RRDS VSAM correspond pratiquement à l'ancienne organisation RELATIVE. Elle consiste à désigner un enregistrement logique de fichier sur mémoire à accès sélectif par le numéro d'enregistrement, compté à partir de un pour le premier enregistrement du fichier (zéro pour l'ancienne organisation Relative OS).

Un fichier Relatif RRDS doit être déclaré, à l'aide du langage AMS, par la commande DEFINE CLUSTER *avec l'attribut NUMBERED.*

Notons toutefois que, pour satisfaire à la norme ANS 74, on peut éviter la déclaration préalable AMS mais ceci au détriment des performances de traitement et de la taille mémoire.

On peut charger un fichier Relatif RRDS en mode SEQUENTIAL ØUTPUT ou DIRECT ØUTPUT.

Le fichier est préformaté en « SLOTS », blocs de longueur fixe, ce qui impose exclusivement des enregistrements de longueur fixe. Chaque SLOT correspond à un numéro d'ordre ; les trous dans la séquence des numéros d'ordre d'enregistrements correspondent donc à des slots vides.

Tout slot vide (à la création ou par suite d'annulation) ne contient que des zéros binaires, ce qui interdit tout accès ultérieur à ces enregistrements annulés.

(1) RRDS = Relative Record Data Set.

FORMAT GÉNÉRAL D'UN PROGRAMME RRDS

Le format d'un programme en Organisation Relative RRDS est le suivant :

```
A       B
  IDENTIFICATIØN DIVISIØN.
            ⋮
  ENVIRØNMENT DIVISIØN.
  CØNFIGURATIØN SECTIØN.
            ⋮
  INPUT-ØUTPUT SECTIØN.
   FILE-CØNTRØL.
        SELECT Nom-de-fichier ASSIGN TØ Nom-de-fichier [Nom-de-fichier]
                                        Système-1      [Système-2]
          ØRGANIZATIØN IS RELATIVE
          [ACCESS MØDE IS { SEQUENTIAL }]
                          { RANDØM     }
                          { DYNAMIC    }
                                  [RELATIVE KEY IS Nom-Donnée-1]
          [PASSWØRD IS Zone Mot-de-Passe]
          [FILE STATUS IS Nom-Donnée-2].
[   I-Ø-CØNTROL.                                                  ]
        [RERUN ØN          Nom-de-fichier de Sauvegarde          ]
                  EVERY n RECØRDS ØF Nom-de-fichier              ]
        [SAME [RECØRD] AREA FØR Nom-de-fichier-2                 ]
                                  [Nom-de-fichier-3]——].          ]
  DATA DIVISIØN.
   FILE SECTIØN.
  FD    Nom-de-fichier [BLØCK CØNTAINS n2 { CHARACTERS }]
                                          { RECØRDS     }
                       [RECØRD CØNTAINS n4 CHARACTERS]
                       LABEL RECØRD IS { STANDARD }
                                       { ØMITTED  }
                       [DATA RECØRD IS Nom-Enregistrement-1
                                       [Nom-Enregistrement-2]].
  01    Nom-Enregistrement-1.
            ⋮
  WØRKING-STØRAGE SECTIØN.
[01     Nom-Donnée-2 PIC 99.]
[01     Zone Mot-de-Passe PIC X (8).]
```

```
A       B
 PRØCEDURE DIVISIØN.
[DECLARATIVES.
 Nom-de-Section SECTIØN. USE AFTER STANDARD {EXCEPTIØN / ERRØR}
                                  PRØCEDURE ØN {Nom-de-fichier / INPUT / ØUTPUT / I-Ø}]
     ØPEN {INPUT / ØUTPUT / I-Ø} Nom-de-fichier.
     [START Nom-de-fichier [KEY IS {EQUAL TØ / = / GREATER THAN / > / NØT LESS THAN / NØT <} Nom-Donnée]
                           [INVALID KEY Ordre Impératif].]
     [READ Nom-de-fichier [NEXT] RECØRD [INTØ Nom-Donnée]
                                  [AT END Ordre Impératif / INVALID KEY Ordre Impératif].
     [{WRITE / REWRITE} Nom-d'Enregistrement [FRØM Nom-de-Zone]
                                  [INVALID KEY Ordre Impératif].]
     [DELETE Nom-de-fichier RECØRD
                                  [INVALID KEY Ordre Impératif].]
     CLØSE Nom-de-fichier [WITH LØCK].
```

LES INSTRUCTIONS COBOL

SELECT Nom-de-fichier ASSIGN TØ Nom-de-fichier Système-1 [Nom-de-fichier Système-2]

Cette clause établit une correspondance entre le Nom-de-fichier désigné en Data Division et le Nom-de-fichier désigné dans les cartes de contrôle JCL du système d'exploitation.

Le Nom-de-fichier Système s'écrit simplement Nom-de-fichier Symbolique de JCL.

Exemple :

SELECT FIRRDS ASSIGN TØ FIRRDJCL

ØRGANIZATIØN IS RELATIVE est indispensable pour désigner un fichier en organisation relative.

$$\left[\text{ACCESS MØDE IS } \left\{\begin{array}{l}\text{SEQUENTIAL}\\ \text{RANDØM}\\ \text{DYNAMIC}\end{array}\right\}\right]$$

Cette clause indique le type d'accès au fichier utilisé dans le programme.

SEQUENTIAL, option par défaut, signifie que le fichier sera traité séquentiellement suivant l'ordre normal des numéros d'ordres croissants.

RANDØM, signifie que le fichier sera traité en accès direct sur le numéro d'ordre spécifié dans la RELATIVE KEY, Nom-Donnée-1.

DYNAMIC, signifie que le fichier sera traité à la fois séquentiellement et en accès-direct.

[RELATIVE KEY IS Nom-Donnée-1]

Nom-Donnée-1, élément simple numérique sans signe, permet de désigner le numéro d'enregistrement qu'on désire atteindre dans le fichier. La RELATIVE KEY ne peut appartenir à l'enregistrement du fichier relatif. En accès séquentiel RELATIVE KEY n'a pas de signification sauf si l'instruction START est utilisée, auquel cas le traitement commence au numéro d'ordre désigné dans Nom-Donnée-1.

[PASSWØRD IS Mot-de-Passe]

Cette clause est associée à la déclaration d'une zone alphanumérique de 8 octets en WØRKING-STØRAGE SECTIØN.

Exemple :

PASSWØRD IS CESAME et 01 CESAME PIC X(8)

Lorsque cette option est programmée, il faut que la zone Mot-de-Passe associée contienne le bon mot-de-passe donnant accès au fichier relatif RRDS au moment de l'exécution du programme.

[FILE STATUS IS Nom-Donnée-2]

Cette instruction désigne la STATUS KEY, zone donnée de deux caractères, qui indique les conditions de réalisation de toute opération d'entrée-sortie sur le fichier correspondant. La STATUS KEY doit être décrite en WØRKING-STØRAGE, sous forme numérique externe.

Exemple :

FILE STATUS IS STATKEY et 01 STATKEY PIC 99.

Voir le paragraphe GESTION DES ERREURS « LA STATUS KEY » au chapitre IV pour plus de détails.

[RERUN ØN~~~~~~~~~~~~~~]
[SAME [RECØRD] AREA~~~] } Voir l'organisation consécutive ESDS

FILE SECTIØN. La description de fichier FD est obligatoire bien que seule la clause LABEL RECØRD soit exigée par COBOL.

$$\left[\text{BLØCK CØNTAINS } n2 \left\{\begin{array}{l}\text{CHARACTERS}\\ \text{RECØRDS}\end{array}\right\}\right]$$

Cette option est inutile en VSAM car la notion de blocs n'a pas de sens.

[RECØRD CØNTAINS *n*4 CHARACTERS]

Cette option est inutile car le compilateur détermine la taille des enregistrements d'après leurs descriptions.

$$\text{LABEL RECØRD IS} \left\{\begin{array}{l}\text{STANDARD}\\ \text{ØMITTED}\end{array}\right\}$$

Cette clause doit toujours être présente bien qu'elle n'ait aucun objet en VSAM puisque tout transite par les catalogues.

[DATA RECØRD IS Nom-Enregistrement-1 [Nom-Enregistrement-2]]

La clause DATA RECØRD n'est plus obligatoire car les noms d'enregistrements sont obligatoirement ceux des descriptions de structures associées au fichier.

$$\text{ØPEN} \left\{\begin{array}{l}\text{INPUT}\\ \text{ØUTPUT}\\ \text{I-Ø}\end{array}\right\} \text{ Nom-de-fichier}$$

INPUT signifie que le fichier est en lecture seule.
ØUTPUT signifie que le fichier est en écriture (création).
I-Ø signifie que le fichier est en mode mise-à-jour-lecture et écriture.

START Nom-de-fichier

Cette instruction ne peut être utilisée qu'en traitement SEQUENTIAL ou DYNAMIC en lecture INPUT ou mise-à-jour I-Ø.

On utilise l'instruction START lorsqu'on désire commencer un traitement séquentiel ailleurs qu'au début du fichier, on désigne alors la clé de départ à l'aide de l'instruction START. On peut aussi spécifier dans l'option KEY que le premier enregistrement devra avoir une clé égale, supérieure ou non-inférieure à la valeur désignée dans Nom-Donnée.

Lorsque la condition n'est pas satisfaite, la condition INVALID KEY

est excitée, ce qui impose l'écriture de cette option pour parer à toute éventualité.

READ Nom-de-fichier [NEXT] RECØRD [INTØ Nom-de-donnée]
[AT END Ordre Impératif]

Ce format est à utiliser en accès séquentiel (ACCESS SEQUENTIAL) ou direct (DYNAMIC) en lecture INPUT ou mise-à-jour I-Ø.

En accès séquentiel on obtient l'enregistrement suivant en séquence. En accès DYNAMIC il faut spécifier l'option NEXT pour atteindre l'enregistrement en séquence, sinon on atteind l'enregistrement dont le numéro d'ordre est désigné par la RELATIVE KEY.

Avant tout ordre de lecture READ le pointeur d'enregistrement en cours (CURRENT RECORD POINTER) doit être convenablement positionné par un ordre ØPEN, START ou READ précédent.

READ Nom-de-fichier RECØRD [INTØ Nom-Donnée]
[INVALID KEY Ordre Impératif]

Ce format est à utiliser en accès direct (ACCESS RANDØM ou ACCESS DYNAMIC).

Avant exécution de ce READ, il faut avoir chargé la RELATIVE KEY de la valeur du numéro d'enregistrement à rechercher car c'est à partir de cette RELATIVE KEY qu'est effectuée la recherche d'enregistrement.

Si l'enregistrement ne peut être trouvé, alors la condition INVALID KEY est activée ainsi que la STATUS KEY, ce qui permet de traiter très complètement cette anomalie.

On pourra aussi omettre la clause INVALID KEY, à condition d'avoir programmé une section DECLARATIVES.

USE AFTER STANDARD { EXCEPTIØN / ERRØR }

En fait, le programmeur a le choix :

. Programmer autant de séquences INVALID KEY qu'il y a de cas.
. Programmer une section Declaratives générale analysant toutes les valeurs possibles de STATUS KEY.

[WRITE Nom-d'Enregistrement [FRØM Nom-de-Zone]
[INVALID KEY Ordre Impératif]]

Cette instruction exige un fichier ouvert en mode ØUTPUT ou I-Ø.

. En accès SEQUENTIAL on écrit successivement les enregistrements à la suite dans l'ordre 1, 2, 3.., Si une RELATIVE KEY est déclarée on recueille le numéro d'ordre de l'enregistrement après chaque ordre WRITE.

. En accès RANDØM ou DYNAMIC, la RELATIVE KEY doit avoir été chargée au préalable du numéro d'ordre de l'enregistrement à écrire.

Comme pour l'ordre READ, on pourra contrôler la bonne exécution de l'ordre WRITE par la clause INVALID KEY ou par une section Declaratives, en conjonction avec la STATUS KEY.

[REWRITE Nom-d'Enregistrement (FRØM Nom-de-zone]
[INVALID KEY Ordre Impératif]]

Cette instruction exige un fichier ouvert en mode I-Ø.

. En accès SEQUENTIAL il est indispensable d'avoir exécuté un ordre READ préalable. L'enregistrement réécrit est alors celui qui vient d'être lu. Notons aussi que l'option INVALID KEY n'a aucun sens dans ce cas.

. En accès RANDØM ou DYNAMIC, la RELATIVE KEY doit avoir été chargée au préalable du numéro d'ordre de l'enregistrement à réécrire. Dans ce cas l'option INVALID KEY a un sens.

Rappelons qu'on ne peut écrire un enregistrement de longueur différente.

[DELETE Nom-de-fichier RECØRD [INVALID KEY Ordre Impératif]]

Cette clause exige un fichier ouvert en mode I-Ø, elle permet de supprimer physiquement un enregistrement. L'espace disque est alors disponible pour un nouvel enregistrement de même numéro d'ordre.

. En accès SEQUENTIAL, comme pour l'ordre REWRITE, il est indispensable d'avoir exécuté un ordre READ préalable. L'enregistrement à supprimer est celui qui vient d'être lu. Ici aussi, l'option INVALID KEY n'a pas de sens.

. En accès RANDØM ou DYNAMIC, la RELATIVE KEY doit avoir été chargée au préalable du numéro d'ordre de l'enregistrement à supprimer.

L'option INVALID KEY a un sens ici.

Rappelons qu'un enregistrement supprimé ne peut plus être atteint puisque le « slot » est mis à zéro binaire.

CLØSE Nom-de-fichier [WITH LØCK]

Tout fichier doit être fermé avant la fin de travail pour permettre au système d'exploitation d'effectuer les contrôles nécessaires.

WITH LØCK est une précaution particulière qui permet d'éviter que le fichier soit à nouveau ouvert dans le même programme.

EXEMPLE DE PROGRAMMATION EN ORGANISATION RELATIVE VSAM-RRDS

Un fichier Relatif VSAM-RRDS peut être créé séquentiellement ou en accès direct.

Création séquentielle

Rappelons qu'en création séquentielle (SEQUENTIAL) les enregistrements sont écrits en séquence suivant la séquence naturelle des numéros d'ordre, sans tenir compte d'éventuelles ruptures.

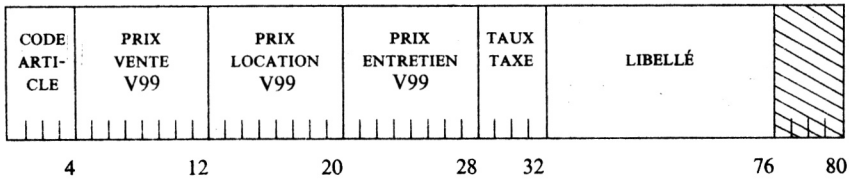
Ceci impose un fichier sans « trous » de numéros d'ordre, ce qui est rarement le cas.

On adoptera donc plutôt la création en accès direct.

Création en accès direct

Les enregistrements sont écrits dans les slots dont les numéros d'ordre sont désignés par la RELATIVE KEY.

Exemple : soit un fichier tarif permettant de déterminer les prix et taux de taxes applicables à des articles désignés par un code numérique à quatre positions.



. Nous commençons par créer, grâce au langage AMS, un cluster VSAM NUMBERED et un fichier de données nommés FIRRDS.

. Puis nous « garnissons » le fichier FIRRDS par le programme COBOL CRERRDS dont le traitement consiste simplement à écrire un enregistrement du fichier FIRRDS pour toute carte lue.

. Par précaution, nous éditons ensuite le contenu du fichier FIRRDS à l'aide de la commande PRINT du langage AMS.

```
//ENOOBONC   JOB   3858,'BONNIN 6577 ',MSGLEVEL=1,CLASS=E
//JOBCAT DD   DSN=SMA.CATAL,DISP=SHR
//STEP00   EXEC   PGM=IDCAMS
//DD1  DD   DISP=OLD,UNIT=3340,VOL=SER=444CCC
//SYSPRINT  DD   SYSOUT=A
//SYSIN    DD   *
       DEFINE  CLUSTER                -
             (NAME(GGDV.RES.FIRRDS)   -
              VOLUME(444CCC)          -
             TRK(3 3)                 -
              FILE  (DD1)             -
              RECORDSIZE (80 80))     -
               DATA                   -
             (NAME(GGDV.RES.FIRRDS.DATA))!
```

```
//STEP01   EXEC    PGM=IDCAMS
//FIRRDS   DD  DSN=GGDV.RES.FIRRDS,DISP=OLD
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
       VERIFY  FILE (FIRRDS);


//STEP02  EXEC COBUCLG,
//  PARM.COB='NODECK,PMAP,VERB,XREF,QUOTE'
//COB.SYSIN  DD  *
001010 IDENTIFICATION DIVISION.
001020 PROGRAM-ID. CRERRDS.
001100 ENVIRONMENT DIVISION.
001110 INPUT-OUTPUT SECTION.
001120 FILE-CONTROL.
001130     SELECT  FIRRDS  ASSIGN TO FIRRDS  ACCESS SEQUENTIAL
001140                     ORGANIZATION RELATIVE  FILE STATUS STATKEY
001150                     RELATIVE KEY CODART.
001160     SELECT  FICART  ASSIGN TO UT-S-FICART.
001200 DATA DIVISION.
001210 FILE SECTION.
001220 FD  FIRRDS  LABEL RECORD STANDARD.
001230 01  ENREG   PIC   X(80).
001250 FD  FICART  LABEL RECORD OMITTED.
001260 01  CARTE.
001270     02  CODART  PIC  9(4).
001280     02  FILLER  PIC  X(76).
001300 WORKING-STORAGE  SECTION.
001310 01  STATKEY PIC  99.
002010 PROCEDURE  DIVISION.
002020 DEBUT.
002030     OPEN INPUT FICART.
002035     OPEN OUTPUT FIRRDS.
           IF STATKEY NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= "
           STATKEY    CALL "ILBOABNO"  GO TO FINTRA.
002040 LECTURE.
002050     READ FICART AT END GO TO FINTRA.
           DISPLAY "CODART = " CODART.
002060     WRITE ENREG  FROM CARTE INVALID KEY  PERFORM ERREUR.
           IF STATKEY NOT = 00 DISPLAY "WRITE IMPOSSIBLE CODART= "
                               CODART "STATUS= " STATKEY GO TO FINTRA.
002070     GO TO LECTURE.
002100 FINTRA.
002110     CLOSE  FICART  FIRRDS.
002120     DISPLAY  "FIN DE TRAVAIL"
002130     STOP  RUN.
002200 ERREUR.
002210     DISPLAY  "ERREUR CARTE = "  CARTE " STATUS KEY = " STATKEY.

//GO.FIRRDS  DD  DSN=GGDV.RES.FIRRDS,DISP=OLD
//GO.SYSOUT  DD  SYSOUT=A
//GO.SYSUDUMP DD SYSOUT=A
//GO.FICART   DD *
```

MISE-A-JOUR DE FICHIER RELATIF VSAM-RRDS

La mise-à-jour du fichier FIRRDS est réalisée par le programme MAJRRDS.

Ce programme consiste à lire des enregistrements mouvement de format identique aux enregistrements du fichier principal, mis à part le code de mise-à-jour placé en dernière position. En cas de modification, les zones non blanches de l'enregistrement mouvement remplacent les zones correspondantes de l'enregistrement principal.

Le principe de traitement est identique aux exemples de traitements de mise-à-jour précédents. A partir de l'indicatif de l'enregistrement mouvement qui détermine le numéro de slot, on lit le fichier permanent FIRRDS.

. Si l'enregistrement du fichier permanent n'est pas trouvé (condition INVALID KEY), le mouvement doit être une création (code MAJ « C »).

. Si l'enregistrement du fichier permanent est trouvé (condition INVALID KEY non excitée), le mouvement est une modification « M » ou une suppression « S ».

En cas d'erreur, nous imprimons la STATUS KEY.

Par précaution, nous plaçons un step VERIFY avant le step de mise-à-jour. A titre de contrôle, nous éditons le contenu du fichier FIRRDS mis-à-jour à l'aide de la commande PRINT du langage AMS.

```
//STEPAA   EXEC  PGM=IDCAMS
//FIRRDS   DD  DSN=GGDV.RES.FIRRDS,DISP=OLD
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
       VERIFY  FILE (FIRRDS)!
```

```
//STEPBB   EXEC  PGM=IDCAMS
//FIRRDS   DD    DSN=GGDV.RES.FIRRDS,DISP=OLD
//SYSPRINT DD    SYSOUT=A
//SYSIN    DD    *
       PRINT                          -
             INFILE  (FIRRDS)         -
             OUTFILE (SYSPRINT)       -
             DUMP!
```

```
//STEP11   EXEC    PGM=IDCAMS
//FIRRDS   DD  DSN=GGDV.RES.FIRRDS,DISP=OLD
//SYSPRINT DD  SYSOUT=A
//SYSIN    DD  *
       VERIFY  FILE (FIRRDS);


//STEP12   EXEC    COBUCLG,
//  PARM.COB='NODECK,PMAP,VERB,XREF,QUOTE'


001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID. MAJRRDS.
001100 ENVIRONMENT DIVISION.
001110 INPUT-OUTPUT SECTION.
001120 FILE-CONTROL.
001130     SELECT  FIRRDS  ASSIGN TO FIRRDS  ACCESS  RANDOM
001140                     ORGANIZATION RELATIVE
001150                     FILE STATUS STATKEY
001160                     RELATIVE KEY MCODART.
001200     SELECT  FICMAJ  ASSIGN TO  UT-S-FICMAJ.
002010 DATA  DIVISION.
002020 FILE  SECTION.
002100 FD  FIRRDS  LABEL RECORD STANDARD.
002110 01  ENREG.
002120     02  CODART    PIC   9(4).
002130     02  PRIVTE    PIC   X(8).
002140     02  PRILOC    PIC   X(8).
002150     02  PRIENT    PIC   X(8).
002160     02  TAXE      PIC   X(4).
002170     02  LIBEL     PIC   X(44).
002180     02  FILLER    PIC   X(4).
002200 FD  FICMAJ  LABEL RECORD STANDARD.
002210 01  CARTE.
002220     02  MCODART   PIC   9(4).
002230     02  MPRIVTE   PIC   X(8).
002240     02  MPRILOC   PIC   X(8).
002250     02  MPRIENT   PIC   X(8).
002260     02  MTAXE     PIC   X(4).
002270     02  MLIBEL    PIC   X(44).
002280     02  FILLER    PIC   X(3).
002290     02  CMAJ      PIC   X.
002300 WORKING-STORAGE SECTION.
002310 01  STATKEY PIC   99.
```

```
003010 PROCEDURE  DIVISION.
003020 DEBUT.
003030     OPEN INPUT  FICMAJ  I-O FIRRDS.
003040     IF STATKEY NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS = "
003050                                 STATKEY    GO TO FINTRA.
003100 LECTURE.
003110     READ FICMAJ AT END GO TO FINTRA.
003140     READ FIRRDS INVALID KEY  GO TO INCONNU.
003142     IF STATKEY NOT = 00 DISPLAY "ERREUR LECTURE  STATUS = "
003144                                 STATKEY    GO TO FINTRA.
003150     IF CMAJ = "M" GO TO MODIFICATION.
003160     IF CMAJ = "S" GO TO SUPPRESSION.
003170     GO TO ERMAJ.
003200 INCONNU.
003202     IF STATKEY NOT = 23 DISPLAY "ERREUR LECTURE  STATUS = "
003204                                 STATKEY    GO TO FINTRA.
003210     IF CMAJ NOT = "C"  GO TO  ERMAJ.
003220     MOVE SPACE TO CMAJ.
003230     WRITE ENREG FROM CARTE INVALID KEY GO TO ERRECR.
003240     GO TO LECTURE.
003300 MODIFICATION.
003310     IF  MPRIVTE NOT = SPACE  MOVE  MPRIVTE  TO  PRIVTE.
003320     IF  MPRILOC NOT = SPACE  MOVE  MPRILOC  TO  PRILOC.
003330     IF  MPRIENT NOT = SPACE  MOVE  MPRIENT  TO  PRIENT.
003340     IF  MTAXE   NOT = SPACE  MOVE  MTAXE    TO  TAXE.
003350     IF  MLIBEL  NOT = SPACE  MOVE  MLIBEL   TO  LIBEL.
003360     REWRITE ENREG INVALID KEY GO TO ERRECR.
003370     GO TO LECTURE.
003400 SUPPRESSION.
003410     DELETE  FIRRDS INVALID KEY GO TO ERRECR.
003420     GO TO LECTURE.
003500 ERMAJ.
003510     DISPLAY "ERREUR MAJ CARTE =" CARTE
003520     GO TO LECTURE.
003600 ERRECR.
003610     DISPLAY "ECRITURE IMPOSSIBLE CODART =" CODART
003620             "STATUS KEY =" STATKEY.
003630     GO TO LECTURE.
003700 FINTRA.
003710     CLOSE  FICMAJ  FIRRDS
003720     DISPLAY "FIN DE MAJRRDS"      STOP RUN.


//GO.SYSUDUMP DD SYSOUT=A
//GO.SYSOUT  DD  SYSOUT=A
//GO.FIRRDS  DD  DSN=GGDV.RES.FIRRDS,DISP=OLD
//GO.FICMAJ    DD    *
```

CHAPITRE VIII

LES INDEX SECONDAIRES

PRINCIPES GÉNÉRAUX

Un index secondaire permet d'atteindre un fichier KSDS suivant un classement différent du classement initial. Par exemple, un fichier de personnel ordonné suivant des matricules croissants comme index principal peut aussi être classé suivant l'ordre alphabétique des noms de personnes comme index secondaire.

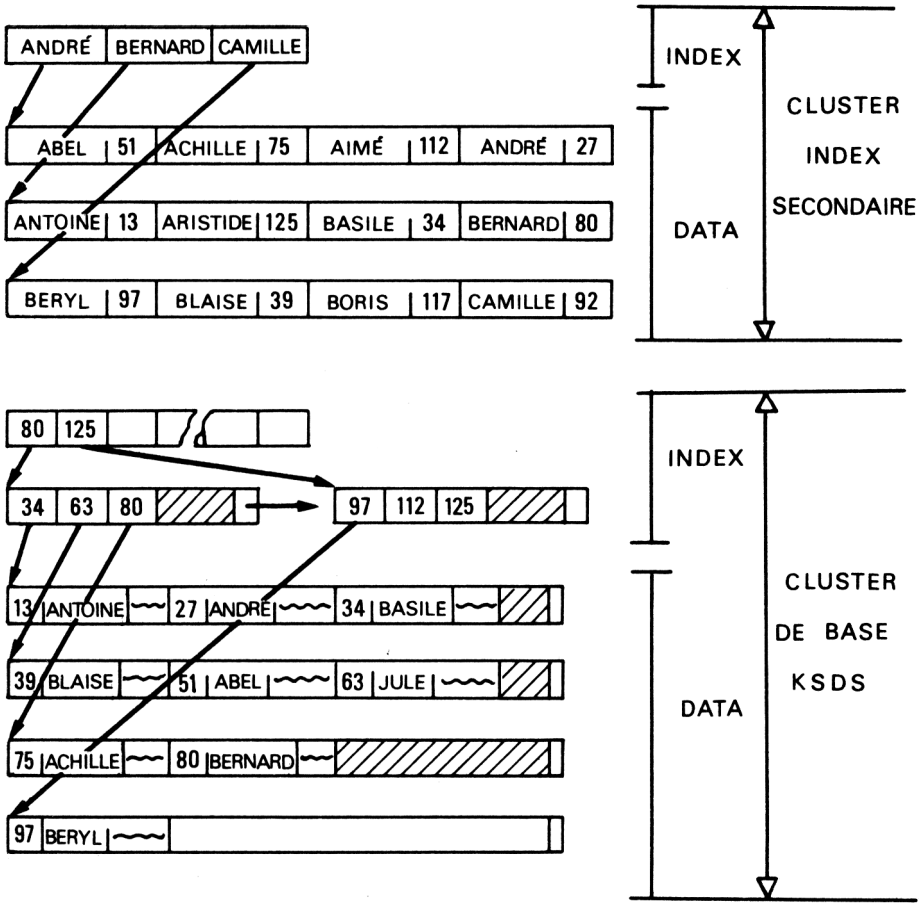
Rappelons qu'un CLUSTER est composé de deux parties — le fichier des données (DATA) et le fichier des index (INDEX), ce dernier virtuel dans le cas d'un fichier ESDS. Le cluster défini pour le fichier de base est appelé CLUSTER DE BASE.

Le cluster index secondaire

Grâce à la commande DEFINE ALTERNATEINDEX du langage spécialisé AMS, on définit un *Cluster Index Secondaire* destiné à recevoir l'organisation de fichier index secondaire.

Comme tout cluster, l'ALTERNATEINDEX est composé de deux parties, une partie INDEX et une partie DATA. La partie INDEX est semblable à celle d'un cluster de base KSDS alors que la partie DATA est composée des clés secondaires ordonnées en séquence associées à des clefs « pointant » sur la partie INDEX du cluster de base dans le cas d'un cluster de base KSDS.

Par exemple (figure ci-après), dans le cas d'un cluster de base KSDS ordonné suivant des numéros de matricules et d'un cluster index secondaire ordonné suivant la séquence alphabétique de prénoms, nous avons



au niveau INDEX du cluster secondaire les prénoms seuls pointant sur une partie DATA qui contient la séquence des prénoms associés au matricule clef primaire correspondant. Ce matricule clef permet à son tour d'atteindre les données du cluster de base via l'index normal du cluster KSDS.

On voit immédiatement que l'accès via l'index secondaire sera obligatoirement moins performant que l'accès via l'index primaire puisqu'il impose un transit par les deux jeux d'index (secondaire puis primaire) pour accéder aux données.

La constitution d'un index secondaire

La constitution physique des fichiers INDEX et DATA de l'index secondaire est réalisée automatiquement par la commande BLDINDEX

du langage AMS à partir du fichier de base qui, bien sûr, doit avoir été constitué au préalable.

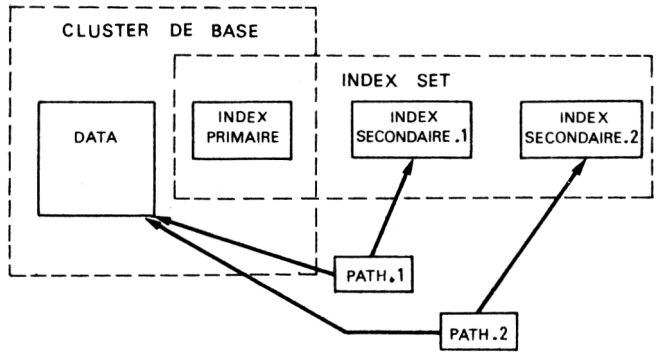
Les INDEX SET-UPGRADE et NOUPGRADE

L'ensemble des parties INDEX (primaire plus secondaires) d'un fichier est appelé INDEX SET, c'est-à-dire ensemble INDEX du fichier.

Lorsqu'on définit un cluster index secondaire avec la commande DEFINE ALTERNATEINDEX, on peut préciser grâce à l'option UPGRADE, si l'index primaire et l'index secondaire doivent être automatiquement maintenus à jour simultanément, que l'accès soit effectué via l'index primaire ou via l'index secondaire.

L'ensemble des index maintenus à jour simultanément est appelé INDEX UPGRADE SET. Inversement, si l'index secondaire est défini avec l'option NOUPGRADE, les modifications du cluster de base ne lui seront pas répercutées — on l'appelle alors INDEX NOUPGRADE SET.

REMARQUE : Rappelons que le cluster de base ne doit pas avoir été déclaré REUSE.



La voie d'accès — PATH

Pour exploiter un fichier via un index secondaire, il est nécessaire de définir une voie d'accès à l'aide de la commande DEFINE PATH du langage AMS qui permet de relier logiquement le fichier DATA de base au cluster Index Secondaire.

C'est le nom de cette voie d'accès qui sera utilisé dans les programmes pour accéder aux enregistrements de base exclusivement via l'index secondaire.

Il est possible de définir plusieurs voies d'accès pour un même ensemble DATA + Index Secondaire, car à chaque PATH est associé un type de

travail, mise-à-jour UPDATE ou non mise-à-jour NOUPDATE. On aura donc même intérêt à utiliser une voie d'accès NOUPDATE pour une lecture exclusive, ce qui allègera d'autant les routines VSAM d'accès et contrôle des données.

Index uniques et index multiples

Il n'est pas toujours possible d'assurer que les index secondaires seront uniques. Par exemple, en indexant sur des noms de familles on a toutes les chances de trouver plusieurs DUPONT, ce qui donnera plusieurs enregistrements ayant la même clé index secondaire. On dit alors être en présence d'index multiples.

Les traitements autorisés seront bien sûr différents selon que l'accès sera réalisé via des index uniques ou multiples, particulièrement pour les ajouts ou suppressions d'enregistrements.

On prendra garde dans le cas d'index multiples, qu'il faut plusieurs pointeurs vers le fichier de base pour une seule clé secondaire, ce qui impose de prévoir des tailles d'enregistrements de la partie DATA de l'index secondaire pouvant contenir un nombre maximum de pointeurs.

Traitements autorisés sur un fichier de base KSDS

Règles de base : Rappelons qu'avec un fichier de base KSDS, tous les traitements sont autorisés — ajout, modification, suppression.

La seule restriction est qu'en mode SEQUENTIAL ØUTPUT seules les créations d'enregistrements sont autorisées.

Clés uniques :

— Les traitements pour un INDEX UPGRADE SET sont identiques à ceux d'un KSDS avec les restrictions suivantes :

. L'écriture, la réécriture ou la suppression ne peuvent se faire que via l'index primaire.
. On ne peut altérer la clef de base d'un enregistrement existant,
. L'ajout d'un enregistrement ne peut créer une clef en double pour le fichier KSDS de base, ni pour l'index secondaire.

— Les traitements pour un INDEX NØUPGRADE SET n'autorisent la suppression ni l'addition d'enregistrements, pas plus que l'altération de clefs.

Clés multiples : L'accès direct ou séquentiel via un index secondaire est possible. En traitement séquentiel, pour une même clé secondaire, les enregistrements sont obtenus dans l'ordre de leur disposition naturelle sur le fichier primaire, que le traitement soit normal ou en lecture arrière.

LES INSTRUCTIONS DU LANGAGE AMS

Définition d'un index secondaire

DEFINE ALTERNATEINDEX
(NAME (Nom-d'Entrée DSNAME du Cluster Index Secondaire)
RELATE (Nom du fichier de Base [/Mot de passe])
KEYS (longueur, position)
[UNIQUEKEY / NØNUNIQUEKEY]
[UPGRADE / NØUPGRADE]

VØLUMES
TRACKS
CYLINDERS
RECØRDS
[ØRDERED/UNØRDERED]
[SPEED/RECØVERY]
[UNIQUE/SUBALLØCATIØN]
[FILE (Nom-DDName)]
[FREESPACE (% par Ci [/ % par CA])]
[CØNTRØLINTERVALSIZE (dimension)]
[KEYRANGES (clé inférieure [/ clé supérieure])]

Paramètres d'Allocation comme définition de Cluster

[MØDEL]
[BUFFERSPACE]

comme définition de catalogue

[WRITECHECK/NØWRITECHECK]
[MASTERPW (Mot-de-Passe)]
[CØNTRØLPW (Mot-de-Passe)]
[UPDATEPW (Mot-de-Passe)]
[READPW (Mot-de-Passe)]
[CØDE (code)]
[ATTEMPTS (Nombre)]
[AUTHØRIZATIØN]
[TØ (Date)/FØR (Durée)]
[ØWNER (propriétaire)]
[SHAREØPTIØNS (Entre régions [/ Entre Systèmes])]
[ERASE/NØERASE]

Paramètres de Protection comme définition de Catalogue

)

[DATA
(NAME (Nom-d'Entrée DSNAME de la partie Données)

Paramètres d'Allocation et de Protection identiques aux précédents

)]

[INDEX
(NAME (Nom d'Entrée DSNAME de la partie Index)

Paramètres d'Allocation et de Protection identiques aux précédents

)]

[CATALØG (Nom du catalogue [/Mot-se-Passe])] ;

Exemple :

```
// STEP01    EXEC PGM = IDCAMS
// SYSPRINT DD   SYSØUT = A
// SYSIN     DD   *
    DEFINE ALTERNATEINDEX                    —
      (NAME (GGDV. RES. ALTIND)              —
            VØLUME (DISGGD)                  —
            TRACKS (10 10)                   —
            KEYS (24 5)                      —
            RELATE (GGDV. RES. FIKDS)        —
            UNIQUEKEY                        —
            UPGRADE                          —
            RECØRDSIZE (80 80))              —
        DATA                                 —
      (NAME (GGDV. RES. ALTIND. DATA))       —
        INDEX                                —
      (NAME (GGDV. RES. ALTIND. INDEX)) ;
```

On définit dans cet exemple un fichier index secondaire GGDV. RES. ALTIND ayant une partie donnée GGDV. RES. ALTIND. DATA et une partie index GGDV. RES. ALTIND. INDEX. Le fichier de base s'appelle GGDV. RES. FIKSDS.
NAME (Nom-d'Entrée DSNAME-~~)

Définir un Index Secondaire revient pratiquement à définir un cluster composé d'une partie données (DATA) et d'une partie INDEX, ce qui autorise à inscrire au catalogue trois noms d'entrées :

— Un nom d'entrée global pour l'Index Secondaire.
— Un nom d'entrée pour la partie DATA.
— Un nom d'entrée pour la partie INDEX.

RELATE (Nom du fichier de base [/Mot-de-Passe])

Le paramètre RELATE désigne le fichier de base pour lequel on définit un index secondaire.

KEYS (longueur position)

Désigne la clef de l'index secondaire.

[UNIQUEKEY
NØNUNIQUEKEY]

Détermine si la clef secondaire est unique (UNIQUEKEY) ou multiple (NØNUNIQUEKEY).

[UPGRADE
NØUPGRADE]

L'option UPGRADE permet de définir si le fichier index secondaire doit faire partie de l'INDEX UPGRADE SET, c'est-à-dire que la clef secondaire soit maintenue à jour avec l'index primaire.

La construction du fichier index secondaire

BLDINDEX

{ INDATASET (Nom-de-Fichier DSNAME [/Mot-de-passe]) (1) }
{ INFILE (Nom-de-Fichier DSNAME [/Mot-de-passe]) }

{ ØUTDATASET (Nom-de-Fichier DSNAME [/Mot-de-passe]) (1) }
{ ØUTFILE (Nom-de-Fichier DDNAME [/Mot-de-passe]) }

[EXTERNALSØRT
INTERNALSØRT]

[WØRKFILES (Nom-de-fichier DDNAME ∮ Nom-de-fichier DDNAME)]
[CATALØG (Nom de catalogue [/Mot-de-passe])]

Exemple :

```
// STEP02   EXEC PGM = IDCAMS
// SYSPRINT DD   SYSØUT = A
// FIKSDS   DD   DSN = GGDV. RES. FIKSDS, DISP = ØLD
// ALTIND   DD   DSN = GGDV. RES. ALTIND, DISP = ØLD
// SYSIN    DD   *
     BLDINDEX                    —
     INFILE (FIKSDS)             —
     ØUTFILE (ALTIND) ;
/*
```

La commande BLDINDEX extrait les clés primaires et secondaires du fichier de base, trie les clés secondaires et charge les informations secondaires dans les parties INDEX et DATA du fichier secondaire.

Avec INDATASET, on indique le Nom de fichier DSNAME au complet, ce qui évite de désigner le fichier par une carte DD du langage JCL de contrôle d'exploitation (1).

Avec INFILE, on indique le Nom de fichier DDNAME, ce qui impose une carte DD dans le JCL pour indiquer le DSNAME.

ØUTDATASET et ØUTFILE désignent le fichier index secondaire.

(1) INDATASET et ØUTDATASET seulement en système MVS.

Avec ØUTDATASET, on indique le Nom de fichier DSNAME au complet, ce qui évite de désigner le fichier par une carte DD du langage JCL de contrôle d'exploitation (1).

Avec ØUTFILE, on indique le Nom de fichier DDNAME, ce qui impose une carte DD dans le JCL pour indiquer le DSNAME.

[EXTERNALSØRT / INTERNALSØRT] ces paramètres permettent d'influencer le déroulement de la phase de tri des index secondaires.

EXTERNALSØRT signifie que le tri sera volumineux et qu'on prévoit l'utilisation de deux fichiers de travail hors mémoire centrale dont les noms de fichiers DDNAME sont habituellement IDCUT1 et IDCUT2.

INTERNALSØRT, option par défaut, signifie que le tri pourra s'effectuer entièrement en mémoire virtuelle et qu'un tri interne est suffisant. A noter qu'on a de toutes façons intérêt à mettre les deux cartes // IDCUT1 et // IDCUT2 dans les cartes de contrôle JCL car, en cas d'insuffisance de mémoire virtuelle, la procédure tente d'utiliser les fichiers externes avant d'abandonner le travail.

[WØRKFILES (Nom-de-fichier DDNAME-1 ⫽ Nombre-de-fichier DDNAME-2)]

Ce paramètre n'est à utiliser que dans le cas où on utilise d'autres noms de fichiers DDNAME que les noms standards IDCUT1 et IDCUT2 pour les fichiers de travail.

[CATALØG (Nom de catalogue [/Mot-de-passe)]

Ce paramètre sert, éventuellement, à désigner le catalogue où sont inscrits les fichiers de travail, s'ils sont catalogués.

La définition d'une voie d'accès — PATH

DEFINE PATH
(NAME (Nom-d'Entrée DSNAME de la voie d'accès)
PATHENTRY (Nom-d'Entrée DSNAME de l'Index Secondaire [/Mot-de-passe])
[UPDATE / NØUPDATE]
[MØDEL] I comme définition de catalogue
[MASTERPW (Mot-de-passe)]
[CØNTRØLPW (Mot-de-passe)]
[UPDATEPW (Mot-de-passe)]
[READPW (Mot de-passe)]
[CØDE (code)
[ATTEMPTS (Nombre)]
[AUTHØRIZATIØN —]
[TØ (Date)/FØR (Durée)]
[ØWNER (propriétaire)]
Paramètres de Protection comme définition de catalogue
)
[CATALØG (Nom de catalogue) [/Mot-de-passe])] ;

(1) Système MVS uniquement.

Exemple :

```
DEFINE PATH                                    —

                (NAME (GGDV. RES. FICBIS)      —
                PATHENTRY (GGDV. RES. ALTIND)  —
                UPDATE) ;
```

Lorsque le cluster index secondaire a été défini par la commande DEFINE ALTERNATEINDEX, puis que le fichier index secondaire a été créé par la commande BLDINDEX, il est possible d'utiliser le cluster index secondaire seul, mais l'intérêt paraît assez mince.

En fait, on utilisera essentiellement l'index secondaire comme voie d'accès secondaire aux données du cluster de base. C'est la commande DEFINE PATH qui permet de créer la voie d'accès en reliant logiquement le cluster de base au cluster index secondaire.

REMARQUE : On peut très bien définir une voie d'accès pour un cluster sans index secondaire, ceci par exemple pour créer une voie d'accès NØUPDATE interdisant la mise-à-jour du fichier. Dans ce cas le paramètre PATHENTRY (voir ci-après) doit désigner le cluster de base et non pas un cluster index secondaire.

NAME (Nom-d'Entrée DSNAME de la voie d'accès).

Le paramètre NAME permet de désigner le nom d'entrée DSNAME du fichier secondaire auquel il sera fait référence dans le programme COBOL. Ne pas oublier que le nom de fichier utilisé dans le programme COBOL est le DDNAME, mais rien n'empêche d'utiliser le même nom comme DSNAME et DDNAME.

PATHENTRY (Nom-d'Entrée DSNAME de l'Index Secondaire [/Mot-de-passe])

Ce paramètre est destiné à désigner le cluster Index secondaire auquel s'applique la voie d'accès qu'on définit ici.

[UPDATE / NØUPDATE] Le paramètre UPDATE, option par défaut, signifie que la voie d'accès définie, autorise la mise-à-jour des informations du fichier de base.

NØUPDATE signifie que les modifications sont interdites.

FORMAT GÉNÉRAL D'UN PROGRAMME ACCÉDANT A DES INDEX SECONDAIRES

```
A       B
 IDENTIFICATIØN DIVISIØN.
 ENVIRØNMENT DIVISIØN.
 CØNFIGURATIØN SECTIØN.
 INPUT-ØUTPUT SECTIØN.
  FILE-CØNTRØL.
       SELECT Nom-de-fichier ASSIGN TØ Nom-de-fichier Système-1 [Nom-de-fichier Système-2]
                      ØRGANIZATIØN IS INDEXED
                      [ACCESS MØDE IS { SEQUENTIAL | RANDØM | DYNAMIC }]
                       RECØRD KEY IS Nom-donnée-2
                       ALTERNATE RECØRD KEY IS Nom-Donnée-4
                                                [WITH DUPLICATES]
                       [PASSWØRD IS Zone Mot-de-passe]
                       [FILE STATUS IS Nom-Donnée-1].
[   I-Ø-CØNTRØL.                                                    ]
[      [RERUN ØN Nom-de-fichier de sauvegarde
                    EVERY n RECØRDS ØF Nom-de-Fichier]             ]
[      [SAME [RECØRD] AREA FØR Nom-de-Fichier-2                    ]
                                  [Nom-de-Fichier-3]---]..         ]
  DATA  DIVISIØN.
   FILE  SECTIØN.
  FD    Nom-de-Fichier LABEL RECØRD IS STANDARD
                      [DATA RECØRD IS Nom-Enregistrement-1
                                    [Nom-Enregistrement-2].]
  01    Nom-Enregistrement-1.
 WØRKING-STØRAGE SECTIØN.
[ 01    Nom-Donnée-1 PIC 99.]
[ 01    Zone Mot-de-Passe PIC X(8).]
```

```
A     B
PRØCEDURE DIVISIØN.
[ DECLARATIVES.
  Nom-de-Section SECTIØN. USE AFTER STANDARD { EXCEPTIØN }
                                             { ERRØR    }
                                  PRØCEDURE ØN { Nom-de-Fichier }
                                               { INPUT        }
                                               { ØUTPUT       }.
                                               { I-Ø          }
                                               { EXTEND       }
  ØPEN { INPUT  }
       { ØUTPUT }
       { I-Ø    } Nom-de-Fichier.
       { EXTEND }
  [ START                [ KEY IS { EQUAL TØ      } Nom-Donnée ]
  Nom-de-fiichier                 {  =             }
                                  { GREATER THAN   }
                                  {  >             }
                                  { NØT LESS THAN  }
                                  { NØT <          }
                  [INVALID KEY Ordre Impératif]. ]
  [ READ Nom-de-fichier [NEXT] RECØRD [INTØ Nom-Donnée-3]
                                      [AT END Ordre Impératif].
  READ Nom-de-fichier RECØRD [INTØ Nom-Donnée-3]
                                      [KEY IS Nom-Donnée-5]
                                      [INVALID KEY Ordre Impératif]. ]
  [ { WRITE   } Nom-d'Enregistrement [FRØM Nom-de-Zone]
    { REWRITE }
                                      [INVALID KEY Ordre-Impératif]. ]
  [ DELETE Nom-de-Fichier RECØRD
                                      [INVALID KEY Ordre-Impératif]. ]
  CLØSE Nom-de-Fichier [WITH LØCK].
```

LES INSTRUCTIONS COBOL

On remarque la similitude de format des programmes en organisation indexée VSAM-KSDS et des programmes accédant à des Index Secondaires à deux exceptions près : la clause ALTERNATE RECØRD KEY du paragraphe FILE-CONTRØL et la clause KEY IS--- de l'instruction READ en PRØCEDURE DIVISIØN.

ALTERNATE RECØRD KEY IS Nom-Donnée-4
[WITH DUPLICATES]

Cette clause permet de désigner une clé secondaire d'accès aux enregistrements. Nom-Donnée-4 doit être décrit comme une zone élémentaire alphanumérique et doit évidemment correspondre à la description réalisée en langage AMS. L'option DUPLICATES est indispensable pour désigner un index multiple.

KEY IS Nom-Donnée-5

Lorsqu'un programme utilise plus d'un index pour accéder aux enregistrements d'un fichier il faut utiliser la clause KEY pour désigner la voie d'accès de l'ordre READ.

Lorsque plusieurs index ont été déclarés et que l'option KEY n'est pas utilisée, l'index primaire est implicitement utilisé.

En mode DYNAMIC et traitement séquentiel, une seule clause KEY initiale est suffisante pour une séquence continue, jusqu'à ce que survienne la nécessité d'une autre séquence.

EXEMPLE DE PROGRAMMATION AVEC INDEX SECONDAIRE UNIQUE

Création de l'Index Secondaire Unique :

— A partir du Cluster de base FIKSVSAM que nous avons créé au chapitre VI nous définissons, à l'aide de la commande DEFINE ALTERNATE INDEX, un cluster index secondaire KSECLIBE ayant pour clé unique le libellé du code paie. Ce cluster est défini UPGRADE, c'est-à-dire que les clés secondaires seront maintenues à jour pour toute modification du cluster de base.

Nous définissons pour ce cluster une partie DATA et une partie INDEX indépendantes.

— Après quoi, la commande BLDINDEX bâtit l'index secondaire en triant les clés contenues dans les enregistrements du fichier de base.

A noter que, s'il existe des clés en double dans le fichier FIKSVSAM, les index correspondants de KSECLIBE ne sont pas créés.

— Enfin, une voie d'accès LIBEPATH est créée pour relier logiquement le cluster de base FIKSVSAM au cluster index secondaire KSECLIBE.

— Par précaution, nous éditons, par la commande PRINT, le contenu du fichier KSECLIBE pour nous assurer que la séquence des clés secondaires est correcte.

```
//STEP11     EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
       DEFINE  ALTERNATEINDEX                -
         (NAME(GGDV.RES.KSECLIBE)            -
          VOLUME(444CCC)                     -
          TRACKS(10 10)                      -
          KEYS(30 4)                         -
          UNIQUEKEY                          -
          RELATE(GGDV.RES.FIKSVSAM)          -
          UPGRADE                            -
          RECORDSIZE(56 56))                 -
             DATA                            -
         (NAME(GGDV.RES.KSECLIBE.DATA))      -
             INDEX                           -
         (NAME(GGDV.RES.KSECLIBE.INDEX));
```

Mise-à-jour de fichier indexé KSDS via un Index Secondaire Unique

— Nous savons qu'il n'est pas possible d'écrire, re-écrire ou supprimer des enregistrements du fichier de base KSDS via l'index secondaire.

— A partir d'un fichier carte de même format que le fichier principal, à l'exception du code de mise-à-jour C, M ou S en position 80 nous désirons mettre à jour notre fichier de rubriques de paie.

— Lorsque le code-paie est présent sur les cartes de mise à jour, la lecture du fichier principal indexé est réalisée via l'index primaire, on peut alors créer, modifier, supprimer.

— Lorsque seul le libellé est présent sur les cartes de mise-à-jour, la lecture du fichier principal indexé est réalisée via l'index secondaire, on ne peut alors que modifier ou supprimer.

```
//STEP12     EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//FIKSVSAM   DD    DSN=GGDV.RES.FIKSVSAM,DISP=OLD
//KSECLIBE   DD    DSN=GGDV.RES.KSECLIBE,DISP=OLD
//SYSIN      DD    *
       BLDINDEX                                  -
         INFILE(FIKSVSAM)                        -
         OUTFILE(KSECLIBE)
       DEFINE  PATH                              -
         (NAME(GGDV.RES.LIBEPATH)                -
          PATHENTRY(GGDV.RES.KSECLIBE))

//STEP13     EXEC  PGM=IDCAMS
//KSECLIBE   DD    DSN=GGDV.RES.KSECLIBE,DISP=OLD
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
       PRINT                                     -
         INFILE(KSECLIBE)                        -
         OUTFILE(SYSPRINT)                       -
         DUMP;

//STEP24 EXEC  COBUCLG,
//  PARM.COB='NODECK,PMAP,VERB,XREF,QUOTE'
//SYSLIB DD DISP=SHR,DSN=SMABTP.MACLIB,VOL=SER=SMAD05,UNIT=3330
//COB.SYSIN  DD    *
```

```
001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID.  MAJALTIN.
001030 ENVIRONMENT  DIVISION.
001040 INPUT-OUTPUT  SECTION.
001050 FILE-CONTROL.
001060     SELECT  FIKSVSAM  ASSIGN TO  FIKSVSAM
001070                       ORGANIZATION  INDEXED  ACCESS IS RANDOM
001080                       RECORD KEY IS CODPAIE
001090                       ALTERNATE RECORD KEY  LIBEL
001100                       FILE STATUS  FISTAT.
001110     SELECT  FICARTE   ASSIGN TO  UT-S-FICARTE.
001200 DATA  DIVISION.
001210 FILE  SECTION.
001220 FD  FIKSVSAM      LABEL RECORD STANDARD
001230                  DATA  RECORD PRINCE.


001240 01  PRINCE.
001250     02  CODPAIE   PIC   9(4).
001260     02  LIBEL     PIC   X(30).
001270     02  IMPUT     PIC   X(6).
001280     02  DATCRE    PIC   X(6).
001290     02  DATFIN    PIC   X(6).
001300     02  DEBCRE    PIC   X.
001310     02  FILLER    PIC   XXX.
002010 FD  FICARTE       LABEL RECORD STANDARD
002020                  DATA  RECORD CARTE.
002030 01  CARTE.
002040     02  MCODPAIE  PIC   9(4).
002050     02  MLIBEL    PIC   X(30).
002060     02  MIMPUT    PIC   X(6).
002070     02  MDATCRE   PIC   X(6).
002080     02  MDATFIN   PIC   X(6).
002090     02  MDECRE    PIC   X.
002100     02  FILLER    PIC   X(26).
002110     02  CODMAJ    PIC   X.
002200 WORKING-STORAGE  SECTION.
002210 01  FISTAT  PIC   99.
003010 PROCEDURE  DIVISION.
003020 DEBUT.
003030     OPEN  INPUT  FICARTE  I-O  FIKSVSAM.
003040     IF FISTAT  NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= "
003050                    FISTAT   GO TO FINTRA.
003100 LECTURE.
003110     READ FICARTE AT END GO TO FINTRA.
003130     IF MCODPAIE NOT = SPACE GO TO PRIMINDEX.
003140     IF MLIBEL NOT = SPACE GO TO SECONDINDEX.
003200 INCONNU.
003210     DISPLAY "CARTE MAJ ERRONNEE = " CARTE.
003220     GO TO LECTURE.
003300 PRIMINDEX.
003310     MOVE MCODPAIE TO CODPAIE.
003320     READ FIKSVSAM  KEY  CODPAIE  INVALID KEY GO TO PRIMINCONN
003325     IF FISTAT NOT = 0 GO TO ERREUR3.
003330     IF CODMAJ = "C" GO TO ERREUR1.
003340     IF CODMAJ = "M" GO TO MODIF1.
003350     IF CODMAJ = "S" GO TO SUPRESS1..
```

```
003400 ERREUR1.
003410     DISPLAY "CODE MAJ ERRONNE = " CARTE.
003420     GO TO LECTURE.
004010 MODIF1.
004020     IF MLIBEL   NOT = SPACE MOVE  MLIBEL  TO  LIBEL.
004030     IF MIMPUT   NOT = SPACE MOVE  MIMPUT  TO  IMPUT.
004040     IF MDATCRE  NOT = SPACE MOVE  MDATCRE TO  DATCRE.
004050     IF MDATFIN  NOT = SPACE MOVE  MDATFIN TO  DATFIN.
004060     IF MDECRE   NOT = SPACE MOVE  MDECRE  TO  DEBCRE.
004070     REWRITE PRINCE INVALID KEY GO TO ERREUR2.
004075     IF FISTAT NOT = 0 GO TO ERREUR3.
004080     GO TO LECTURE.
004100 SUPRESS1.
004110     DELETE FIKSVSAM  INVALID KEY GO TO ERREUR2.
004115     IF FISTAT NOT = 0 GO TO ERREUR3.
004120     GO TO LECTURE.
004200 ERREUR2.
004210     DISPLAY "ACCES IMPOSSIBLE CARTE = " CARTE
004220             " STATUS = " FISTAT.
004230     GO TO LECTURE.
004300 PRIMINCONNU.
004310     IF FISTAT NOT = 23 GO TO ERREUR2.
004320     WRITE PRINCE FROM CARTE  INVALID KEY GO TO ERREUR2.
004325     IF FISTAT NOT = 0 GO TO ERREUR3.
004330     GO TO LECTURE.
005010 SECONDINDEX.
005020     MOVE MLIBEL TO LIBEL.
005030     READ FIKSVSAM  KEY LIBEL INVALID KEY GO TO INCONNU.
           IF FISTAT NOT = 0 GO TO ERREUR3.
005040     IF CODMAJ = "M" GO TO MODIF2.
005050     IF CODMAJ = "S" GO TO SUPPRESS2.
005060     GO TO ERREUR1.
005100 MODIF2.
005110     IF  MIMPUT    NOT = SPACE MOVE   MIMPUT  TO IMPUT.
005120     IF  MDATCRE   NOT = SPACE MOVE   MDATCRE TO DATCRE.
005130     IF  MDATFIN   NOT = SPACE MOVE   MDATFIN TO DATFIN.
005140     IF  MDECRE    NOT = SPACE MOVE   MDECRE  TO DEBCRE.
005150     REWRITE PRINCE INVALID KEY GO TO ERREUR2.
005155     IF FISTAT NOT = 0 GO TO ERREUR3.
005160     GO TO LECTURE.
005200 SUPPRESS2.
005210     DELETE FIKSVSAM  INVALID KEY GO TO ERREUR2.
005215     IF FISTAT NOT = 0 GO TO ERREUR3.
005220     GO TO LECTURE.
005300 ERREUR3.
005310     DISPLAY "I/O ERREUR - STATUS = " FISTAT GO TO FINTRA.
005400 FINTRA.
005410     CLOSE FIKSVSAM FICARTE DISPLAY "FIN DE MAJALTIN" STOP RUN.


//GO.SYSOUT     DD  SYSOUT=A
//GO.SYSUDUMP   DD  SYSOUT=A
//GO.FIXSVSAM   DD  DSN=GGDV.RES.FIKSVSAM,DISP=OLD
//GO.FICARTE    DD  *
```

EXEMPLE DE PROGRAMMATION AVEC INDEX SECONDAIRE MULTIPLE

Création d'un cluster index secondaire multiple :

— A partir du même fichier FIKSVSAM nous définissons, à l'aide de la commande DEFINE ALTERNATEINDEX, un cluster index secondaire KSECIMPU ayant pour clé multiple le code imputation, longueur (6) et position 34 (comptée à partir de zéro).

Ce cluster est défini NOUPGRADE puisque les clés ne sont pas uniques. Nous avons prévu une taille d'enregistrement (RECORDSIZE) suffisamment importante pour contenir une clé secondaire et plusieurs clés primaires.

— Après quoi, la commande BLDINDEX bâtit l'index secondaire en triant les clés contenues dans les enregistrements du fichier de base.

— Enfin, une voie d'accès IMPATH est créée pour relier logiquement le cluster de base FIKSVSAM au cluster index secondaire KSECIMPU.

— Par précaution, nous éditons par la commande PRINT le contenu du fichier KSECIMPU pour nous assurer que la séquence des clés secondaires est correcte.

Édition de Fichier Indexé VSAM via un Index Secondaire multiple :

L'intérêt des index secondaires multiples est de permettre aisément l'édition d'un fichier indexé sur une clé secondaire multiple.

Le programme EDITALTE permet d'éditer la liste des codes paies par imputation avec pour chaque code paie, le libellé, les dates de création et annulation et le code Débit ou Crédit.

```
//JOBCAT DD   DSN=SMA.CATAL,DISP=SHR
//STEP51     EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
       DEFINE  ALTERNATEINDEX                -
         (NAME(GGDV.RES.KSECIMPU)            -
          VOLUME(444CCC)                     -
          TRACKS(10 10)                      -
          KEYS(6 34)                         -
          RELATE(GGDV.RES.FIKSVSAM)          -
          NONUNIQUEKEY                       -
          NOUPGRADE                          -
          RECORDSIZE(200 200))               -
             DATA                            -
         (NAME(GGDV.RES.KSECIMPU.DATA))      -
             INDEX                           -
         (NAME(GGDV.RES.KSECIMPU.INDEX))!


//STEP52     EXEC  PGM=IDCAMS
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
       BLDINDEX                                 -
          OUTDATASET(GGDV.RES.KSECIMPU)
       DEFINE  PATH                             -
          INDATASET(GGDV.RES.FIKSVAM)           -
         (NAME(GGDV.RES.IMPATH)                 -
          PATHENTRY(GGDV.RES.KSECIMPU)          -
          NOUPDATE)


//STEP53     EXEC  PGM=IDCAMS
//KSECIMPU  DD  DSN=GGDV.RES.KSECIMPU,DISP=OLD
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
       PRINT                                    -
          INFILE(KSECIMPU)                      -
          OUTFILE(SYSPRINT)                     -
          DUMP!


//STEP60  EXEC COBUCLG,
//  PARM.COB='NODECK,PMAP,VERB,XREF,QUOTE
//COB.SYSIN  DD    *
```

```
001010 IDENTIFICATION  DIVISION.
001020 PROGRAM-ID.  EDITALTE.
001030 ENVIRONMENT  DIVISION.
001040 INPUT-OUTPUT  SECTION.
001050 FILE-CONTROL.
001060     SELECT  FIKSVSAM  ASSIGN TO  FIKSVSAM
001070                       ORGANIZATION INDEXED  ACCESS IS DYNAMIC
001080                       RECORD KEY IS CODPAIE
001090                       ALTERNATE RECORD KEY IMPUT WITH DUPLICATES
001100                       FILE STATUS FISTAT.
001110     SELECT  FICHEDIT  ASSIGN TO  UT-S-FICHEDIT.
001200 DATA  DIVISION.
001210 FILE  SECTION.
001220 FD  FIKSVSAM       LABEL RECORD STANDARD
001230                   DATA  RECORD PRINCE.
001240 01  PRINCE.
001250     02  CODPAIE   PIC   9(4).
001260     02  LIBEL     PIC   X(30).
001270     02  IMPUT     PIC   X(6).
001280     02  DATCRE    PIC   X(6).
001290     02  DATFIN    PIC   X(6).
001300     02  DEBCRE    PIC   X.
001310     02  FILLER    PIC   XXX.
002010 FD  FICHEDIT       LABEL RECORD OMITTED
002020                   DATA  RECORD LIGNE.
002030 01  LIGNE.
002040     02  SAUT      PIC   X.
002050     02  FILLER    PIC   X(132).
002100 WORKING-STORAGE  SECTION.
002110 01  FISTAT        PIC   99.
002120 01  RESIMPUT      PIC   X(6)  VALUE SPACE.
002200 01  LITIT.
002210     02  FILLER    PIC   X(28) VALUE "  IMPUTATION  *  CODE PAIE".
002220     02  FILLER    PIC   X(35) VALUE "*              LIBELLE".
002230     02  FILLER    PIC   X(26) VALUE "*  CREATION  * ANNULATION".
002240     02  FILLER    PIC   X(43) VALUE "*  D/C  *".
002300 01  LIDET.
002310     02  FILLER    PIC   XX    VALUE  SPACE.
002320     02  EIMPUT    PIC   X(6).
002330     02  FILLER    PIC   X(10) VALUE  "    *".
002340     02  ECODPAIE  PIC   9(4).
002350     02  FILLER    PIC   X(7)  VALUE  "    *".
002360     02  ELIBEL    PIC   X(30).
002370     02  FILLER    PIC   X(5)  VALUE  "  *".
002380     02  EDATCRE   PIC   XXBXXBXX.
002390     02  FILLER    PIC   X(5)  VALUE  "  *".
002400     02  EDATFIN   PIC   XXBXXBXX.
002410     02  FILLER    PIC   X(6)  VALUE  "  *".
002420     02  EDEBCRE   PIC   X.
002430     02  FILLER    PIC   X(40) VALUE  "   *".
```

```
003010 PROCEDURE  DIVISION.
003020 DEBUT.
003030     OPEN  INPUT FIKSVSAM  OUTPUT FICHEDIT.
003040     IF FISTAT  NOT = 00 DISPLAY "OPEN IMPOSSIBLE STATUS= " FISTAT
003050                         CLOSE FIKSVSAM FICHEDIT    STOP RUN.
003100 PRELEC.
003105     MOVE SPACE TO IMPUT.
003107     READ FIKSVSAM KEY IMPUT.
003110     READ FIKSVSAM  NEXT     AT END CLOSE FIKSVSAM FICHEDIT
003120         DISPLAY "ERREUR - FICHIER VIDE - STATUS= " FISTAT
003130         STOP RUN.
003200 EDITIT.
003210     WRITE LIGNE FROM LITIT AFTER 1
003220     MOVE IMPUT TO RESIMPUT EIMPUT.
003300 EDIDET.
003310     MOVE CODPAIE  TO  ECODPAIE     MOVE  DATCRE  TO  EDATCRE
003320     MOVE LIBEL    TO  ELIBEL       MOVE  DATFIN  TO  EDATFIN
003330     MOVE DEBCRE   TO  EDEBCRE
003340     WRITE LIGNE FROM LIDET AFTER 1.
003350     READ FIKSVSAM  NEXT     AT END CLOSE FIKSVSAM FICHEDIT
003360          DISPLAY  "FIN DE EDITALTE"  STOP RUN.
003370     IF FISTAT NOT = 00  DISPLAY "I/O ERREUR STATUS = " FISTAT
003380                         STOP RUN.
003390     IF IMPUT NOT = RESIMPUT  GO TO EDITIT.
003400     MOVE SPACE TO EIMPUT  GO TO EDIDET.


//GO.SYSPRINT   DD SYSOUT=A
//GO.SYSOUT     DD  SYSOUT=A
//GO.SYSUDUMP   DD  SYSOUT=A
//GO.FIKSVSAM   DD  DSN=GGDV.RES.FIKSVSAM,DISP=OLD
//GO.FICHEDIT   DD  SYSOUT=A
```

CHAPITRE IX

LA GESTION DES TABLES

Il était déjà très facile d'utiliser des tables avant l'apparition du COBOL A. N. S. mais ce dernier offre maintenant en plus un support réel de gestion des tables.

Nous ne reviendrons pas en détail sur les principes de base des tables (1) mais rappelons qu'une table est un groupe de données composé d'éléments simples ayant tous des attributs identiques.

Chacun des éléments de la table est désigné à l'aide d'un numéro d'ordre appelé « indice » sous la forme ELEMENT (indice).

L'indice peut être utilisé sous forme d'un nombre entier, par exemple ELEMENT (5) pour désigner le cinquième élément d'une table.

L'indice peut aussi être utilisé sous forme d'un Nom-Donnée, par exemple SALAIRE (ECHELLE) avec ECHELLE prenant exclusivement des valeurs entières.

Puisque COBOL autorise jusque trois indices pour une table, nous pouvons écrire dans le cas le plus général : ELEMENT (indice-1, indice-2, indice-3).

PRINCIPES GÉNÉRAUX DES INDEX

Le COBOL A. N. S. offre la possibilité de remplacer les indices par des INDEX jouant pratiquement le même rôle mais implicitement réservés par COBOL lors de la déclaration de la table.

Par exemple :

```
A       B
01      TABLE.
        02  TABLEAU ØCCURS 2 INDEXED BY IND1.
            04 VECTEUR ØCCURS 2 INDEXED BY IND2.
            06 ELEMENT ØCCURS 3 INDEXED BY IND3
                                        PIC S9(8) CØMP.
```

(1) Voir l'ouvrage précédent « LE COBOL A. N. S. » du même auteur.

suffit à déclarer une table et les trois indices IND1, IND2, IND3 sous forme de quantités binaires sur quatre octets.

L'indexage direct :

Il est alors possible de désigner directement les éléments de la table sous la forme ELEMENT (IND1, IND2, IND3) ; ce qui n'apporte en fait rien de nouveau par rapport aux indices.

L'indexage indirect :

Mais le COBOL A. N. S. apporte la possibilité de réaliser un indexage indirect de la forme : ELEMENT (IND1 ± *n*1, IND2 ± *n*2, IND3 ± *n*3) où *n*1, *n*2, *n*3 désignent des nombres entiers d'éléments dont il faut se déplacer par rapport aux bases IND1, IND2, IND3.

Par exemple dans la table ci-dessous, si IND1 = 1 alors ELEMENT (IND1 + 2) désigne le troisième élément de la table.

	1	2	3	4	5
TABLE :	Quantité A	Quantité B	Quantité C	Quantité D	Quantité E

REMARQUE. A première vue, index et indices paraissent jouer exactement le même rôle, mais en fait la différence essentielle réside dans le contenu des index et indices. Le contenu de l'indice est un nombre entier désignant le numéro d'ordre de l'élément compté à partir de 1 pour le premier ; par contre, le contenu de l'index est le nombre d'octets désignant le déplacement de l'élément par rapport au début de la table, d'où une liaison étroite entre l'index et sa table.

Par exemple avec des éléments de 8 octets de long, on aurait pour le cinquième élément la valeur 4 dans l'indice et la valeur (5 − 1) * 8 = 32 dans l'index. On ne pourra donc mettre directement une valeur origine dans un index par l'ordre MØVE puisqu'une conversion est nécessaire, ce qui conduit à utiliser un nouvel ordre, spécialement créé pour la gestion des tables, l'ordre SET.

Par exemple SET IND1 TØ 1 positionne l'index sur la première occurence des éléments de la table.

Enfin nous verrons que les recherches et manipulations d'éléments de tables ne seront plus effectuées par le verbe PERFØRM mais par le verbe SEARCH (recherche) qui a sensiblement les mêmes origines.

LES VERBES DE GESTION DES TABLES

En premier lieu, il convient de décrire la table en DATA DIVISIØN à l'aide de la clause ØCCURS qui reçoit quelques ajouts.

Nom-Donnée-Elément ØCCURS Nombre-Entier TIMES

[{ ASCENDING / DESCENDING } KEY IS Nom-Donnée-2 [Nom-Donnée-3] ...]

[INDEXED BY Nom-Index-1 [Nom-Index-2] .....]

Nous ne reviendrons pas sur la clause ØCCURS proprement dite et ses deux variantes (1).

ØCCURS Nombre-Entier-1 TØ Nombre-Entier-2 TIMES
DEPENDING ØN Nom-Donnée

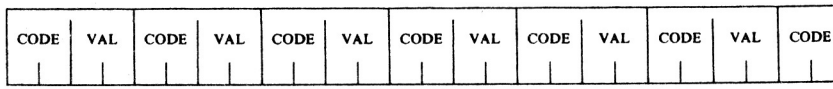
ou

ØCCURS Nombre-Entier-2 TIMES [DEPENDING ØN Nom-Donnée]

[{ ASCENDING / DESCENDING } KEY IS Nom-Donnée-2 [Nom-Donnée-3] ...]

Cette clause spécifie l'argument de classement de la table de manière à préparer les recherches futures par l'ordre SEARCH.

Par exemple, une table composée de codes et valeurs correspondantes, les codes étant classés suivant les valeurs croissantes



sera décrite de la manière suivante :

```
A       B
01      TABLE.
          02  ELEMENT ØCCURS 100 TIMES ASCENDING KEY CØD.
              04  CØD PIC XX.
              04  VALEUR PIC S9(4) CØMP.
```

Il peut éventuellement y avoir plusieurs arguments de classements : Nom-Donnée-2, Nom-Donnée-3, etc... qui devront être désignés dans l'ordre décroissant du classement : majeur, inter, mineur... Un nombre maximum de 12 arguments de classement peuvent être désignés pour un élément.

Les arguments doivent être situés à une adresse fixe par rapport au début de l'élément dont ils sont les clefs et ne peuvent contenir eux-mêmes une clause ØCCURS.

(1) Voir l'ouvrage précédent « LE COBOL A. N. S. » du même auteur.

Par exemple

```
A       B
01      ZØNE.
        02  ELEM ØCCURS 12 ASCENDING KEY IS CUMUL.
            04 CØD PIC XX.
            04 VAL PIC S9(4) CØMP.
        02  CUMUL PIC S9(4) CØMP.
```

Cette description est erronée car CUMUL, zone extérieure à ELEM ne peut être un argument de classement ; le seul argument possible est CØD.

Autre exemple :

```
A       B
01      ZØNE.
        02  DEPARTEMENT ØCCURS 100 ASCENDING KEY IS CØDEP.
            04 NBRE PIC 99.
            04 VILLE ØCCURS 100 DEPENDING
                                NBRE ASCENDING KEY IS CØDVIL.
               06 CØDEP PIC 99.
               06 CØDVIL PIC 999.
               06 HABIT PIC 9(9) CØMP-3.
```

CØDEP ne peut être un argument KEY pour DEPARTEMENT car CØDEP est inclus dans une clause ØCCURS donc situé à une adresse variable par rapport à chaque groupe DEPARTEMENT.

Il aurait été préférable d'écrire :

```
A       B
01      ZØNE.
        02  DEPARTEMENT ØCCURS 100 ASCENDING KEY IS CØDEP.
            04  CØDEP PIC 99.
            04  NBRE  PIC 99.
            04  VILLE ØCCURS 100 DEPENDING
                                NBRE ASCENDING KEY IS CØDVIL.
               06  CØDVIL PIC 999.
               06  HABIT  PIC 9(9) CØMP-3.
```

[INDEXED BY Nom-Index-1 [Nom-Index-2] ....]

Cette clause sert à désigner les index qui seront utilisés dans la PRØCEDURE DIVISIØN pour qualifier l'élément.

Par exemple, le fait d'écrire : 02 ELEM ØCCURS 12 INDEXED BY IND-1 IND-2 IND-3 suppose que seuls les index IND-1, IND-2 et IND-3 seront utilisés avec ELEM. Nous ne pourrons aussi désigner qu'un maximum de 12 index par élément.

Comme nous l'avons vu plus haut, les index sont automatiquement réservés par cette clause sous forme de quantités binaires sur quatre octets.

Les index ne peuvent être utilisés qu'après avoir été initialisés par l'ordre SET.

SET Nom-Index-1 [Nom-Index-2] TØ { Nombre Entier / Nom-Index-3 }

En français : METTRE Nom-Index-1 [Nom-Index-2]
A LA VALEUR CØRRESPØNDANT A { Nombre-Entier / Nom-Index-3 }

Nombre entier représente un numéro d'occurence de l'élément qu'on désire repérer.

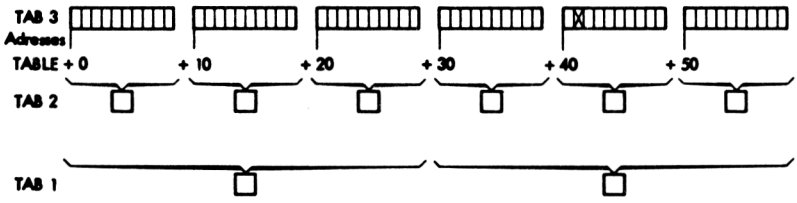
Par exemple : SET INDEX1 TØ 5 signifie que nous voulons atteindre le cinquième élément de la table. Ce n'est pas la valeur 5 que nous retrouverons dans INDEX1 mais l'adresse correspondant au déplacement de l'élément par rapport au début de table suivant la formule INDEX1 = (occurence – 1) * longueur de l'élément.

Lorsque Nom-Index-3 est utilisé, il y a ajustement du contenu de Nom-Index-1, Nom-Index-2... sur Nom-Index-3 par le compilateur COBOL qui traduit le contenu de Nom-Index-3 en occurence avant de calculer les valeurs de Index-1 et Index-2.

Exemple :

```
A       B
01      TABLE.
        02  TAB1 ØCCURS 2 INDEXED BY IND1.
            04 TAB2 ØCCURS 3 INDEXED BY IND2.
               06  TAB3  ØCCURS  10  INDEXED  BY  IND3  PIC  X.
```

Cette table peut être schématisée de la manière suivante :



Nous pouvons expliciter la manière de désigner TAB3 (2, 2, 2)

SET IND3 TØ 2 ⇒ donne IND3 = (2 – 1) * 1 = 1
SET IND1 IND2 TØ IND3 ⇒ donne IND2 = (2 – 1) * 10 = 10
et IND3 = (2 – 1) * 30 = 30

d'où TAB3 (IND1, IND2, IND3) désigne l'adresse 1 + 10 + 30 = 41 ce qui est conforme à ce que nous désirions.

SET Nom-Index-1 [Nom-Index-2] { UP BY / DØWN BY } { Nombre-Entier / Nom-Index-3 }

Cette variante permet de faire croître (UP BY) ou décroître (DØWN BY) le contenu de Index-1, Index-2 en fonction de Nombre-Entier

représentant une occurence, ou de Nom-Index-3 traduit en nombre d'occurences.

Cela revient à appliquer la formule INDEX1 = INDEX1 ± (Nombre-Entier * Longueur de l'élément).

A noter qu'il n'est pas obligatoire que Index-3 corresponde à la même table que Index-1 ou Index-2 car on traduit son contenu en nombre d'occurences.

Le cas échéant, on pourra désirer sauvegarder une valeur d'index dans une zone de réserve, on utilisera alors un Nom-Donnée d'usage index.

Nom-Donnée [USAGE IS] INDEX.

Cette clause est utilisée à n'importe quel niveau de Data Division.

Il n'est pas utile de déclarer de PICTURE ni d'USAGE car COBOL réserve automatiquement une quantité binaire sur quatre octets.

Exemple :

```
|A      |B
|77     |ZØRES USAGE IS INDEX.
```

Lorsque cette clause est spécifiée pour une zone groupe, ce sont toutes les zones élémentaires du groupe qui sont considérées d'usage index.

Ces Noms d'usage index seront utilisés comme les index exclusivement avec les ordres SET et SEARCH. On fera toutefois attention à l'utilisation des Noms Données d'usage index lorsqu'on y sauvegarde une valeur d'index.

Par exemple en reprenant l'exemple des index.

```
|A      |B
|01     |TABLE.
|       |02  TAB1 ØCCURS 2 INDEXED BY IND1.
|       |    04  TAB2 ØCCURS 3 INDEXED BY IND2.
|       |        06  TAB3 ØCCURS 10 INDEXED BY IND3 PIC X.
```

avec

```
SET IND3 TØ 2
SET IND1 IND2 TØ IND3
```

nous obtenons un déplacement d'adresse + 41 mais si nous écrivons

```
SET IND3   TØ 2
SET ZØRES TØ IND3
SET IND1   IND2 TØ ZØRES
```

nous ne pourrons obtenir la même adresse résultante car IND1, IND2, IND3 sont de véritables INDEX dont on traduit implicitement le contenu en occurences pour tout ordre SET alors que ZØRES ne correspondant à aucune table, son contenu est simplement mouvementé.

En effet

```
SET IND3   TØ 2              ⇒ donne IND3   = (2 − 1) * 1 = 1
SET ZØRES TØ IND3            ⇒ donne ZØRES = 1
SET IND1 IND2 TØ ZØRES ⇒ donne IND1   = IND2 = 1
```

d'où TAB3 (IND1, IND2, IND3) correspond alors au déplacement 1 + 1 + 1 = 3 qui n'est pas du tout ce que nous attendions.

L'ordre SEARCH

Le verbe SEARCH permet de rechercher en table un élément qui satisfasse à une ou plusieurs conditions données.

Les conditions mettront généralement en œuvre des comparaisons de valeurs d'index aussi le programmeur devra-t-il constamment garder à l'esprit que ce ne sont pas les contenus des index qui seront comparés mais leur équivalent en nombre d'occurences.

En conséquence :

— La comparaison de deux index, même s'ils appartiennent à des tables différentes, revient à comparer leurs équivalents en nombre d'occurences.

— La comparaison d'un index et d'un Nom-Donnée revient à comparer le contenu de la zone Nom-Donnée avec l'équivalent en nombre d'occurences du contenu de l'index (le contenu de la zone Nom-Donnée devra donc être obligatoirement un nombre entier).

Mais les Noms-Données d'usage-index font exception à la règle :

— La comparaison d'un index avec un Nom-Donnée d'usage index sera faite sur les contenus réels des deux zones car, nous l'avons déjà vu, on ne peut traduire un Nom-Donnée d'usage index en nombre d'occurences puisqu'il n'appartient en fait à aucune table. On doit donc recommander de n'utiliser les Noms-Données d'usage index qu'avec précaution.

Notons aussi qu'on ne peut comparer un Nom-Donnée d'usage index qu'avec un index ou un autre Nom-Donnée d'usage index.

La table ci-dessous résume les différents cas de comparaisons autorisés :

	Index	Nom-Donnée	Littéral Numérique	Nom-Donnée d'Usage Index
INDEX	Comparaison des équivalents en nombre d'occurences.	Comparaison du contenu de la zone Nom-Donnée avec l'équivalent du contenu de l'index en nombre d'occurences.	Comparaison du littéral avec l'équivalent du contenu de l'index en nombre d'occurences.	Comparaison directe des contenus sans conversion.

Le verbe SEARCH ressemble beaucoup au PERFØRM VARYING mais présente le désavantage de ne pas permettre d'explorer l'ensemble d'une table à plusieurs indices (deux ou trois) en une seule boucle comme il est possible de le faire avec le PERFØRM à deux ou trois indices. Une autre différence est que nous devrons mettre une valeur initiale à chaque index par l'ordre SET avant tout verbe SEARCH.

Premier Format du Verbe SEARCH.

SEARCH Nom-Élément-1

[VARYING { index-1 / Nom-Donnée-2 }] [AT END Ordre-Impératif-1]

WHEN Condition-1 { Ordre Impératif-2 / NEXT SENTENCE }

[WHEN Condition-2 { Ordre Impératif-3 / NEXT SENTENCE }] ...

En français :

EXPLORER élément-1 [EN FAISANT VARIER { index-1 / Nom-Donnée-2 }]

[EN FIN DE TABLE faire Ordre-Impératif-1]

si la Condition-1 est satisfaite faire { ordre-impératif-2 / PHRASE SUIVANTE }

[Si la Condition-2 est satisfaite faire { Ordre-Impératif-3 / PHRASE SUIVANTE }] ...

Nom-Elément-1 doit appartenir à une table c'est-à-dire être lié à une clause ØCCURS et, contrairement aux usages, on ne doit pas associer d'indices ou d'index à ce Nom-Elément ; par exemple, au lieu de TABLEAU (J) on écrira simplement SEARCH TABLEAU.

[VARYING { Index-1 / Nom-Donnée-2 }]

— Si Index-1 a été déclaré être un index pour Elément-1, on fait varier index-1 à chaque boucle à partir d'une valeur initiale placée au préalable par l'ordre SET.

— Si index-1 n'est pas un index pour Elément-1, il y a progression à la fois de index-1 et du premier index déclaré pour Elément-1. On voit donc qu'on peut éventuellement faire progresser simultanément l'index d'une deuxième table pendant l'exploration de la table correspondant à Elément-1.

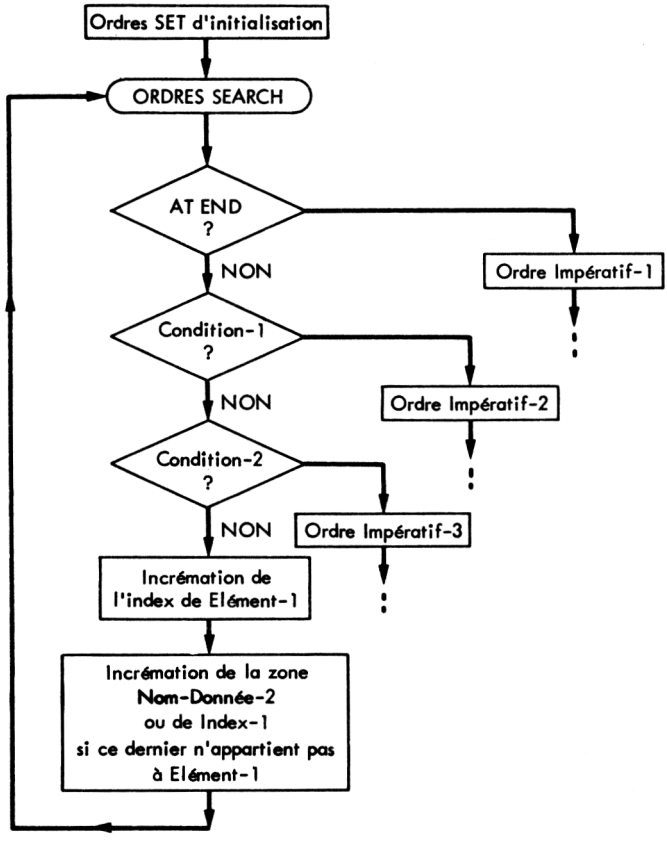
— Nom-Donnée-2 sera un Nom-Donnée d'Usage Index ou une zone de donnée numérique entière dont le contenu sera incrémenté conjointement, et de la même valeur, que l'index allié à Elément-1 à chaque boucle donc à chaque occurence de l'élément-1.

WHEN Condition { Ordre Impératif / NEXT SENTENCE }

La condition peut être quelconque et ne devra pas obligatoirement faire intervenir des index ou arguments de classement (KEY) comme ce sera le cas dans le second format de l'ordre SEARCH.

Avant tout ordre SEARCH les différents index (ou Nom-Donnée-2) utilisés devront être initialisés par une instruction SET et c'est à partir de cette valeur initiale que débute la boucle SEARCH ; l'incrémentation des index n'ayant lieu qu'en fin de boucle.

A chaque boucle, COBOL évalue tout d'abord la condition AT END de fin de table ; si celle-ci n'est pas satisfaite, COBOL évalue la première condition WHEN puis la seconde condition WHEN (si celle-ci est présente). Si aucune des conditions n'est satisfaite, COBOL fait alors progresser l'index correspondant à élément-1 et éventuellement index-1 si celui-ci ne correspond pas à élément-1 ; enfin il y a retour en début de boucle.



REMARQUE : Si l'ordre impératif-1, -2 ou -3 ne provoque pas un débranchement, on passe tout simplement à l'ordre suivant comme toujours en COBOL.

Exemple : Soit une table décrite comme suit :

```
A       B
01      TABCØDPAIE.
        02  ELEMENT ØCCURS 100 INDEXED BY IND1.
            04 CØD PIC XX.
            04 VALEUR PIC S9(4) CØMP.
```

Pour rechercher dans cette table une valeur correspondant à un code CØDPAIE, on écrira :

```
A       B
        SET IND1 TØ 1
        SEARCH ELEMENT
                AT END GØ TØ ERREUR1
                WHEN CØD (IND1) = CØDPAIE GØ TØ CALCUL
                WHEN CØD (IND1) = "99" GØ TØ ERREUR1.
```

Lorsque le code est trouvé il y a débranchement à CALCUL ; lorsque le code n'est pas trouvé (la fin de table ou le code 99 sont atteints) il y a débranchement à ERREUR1.

Second format du Verbe SEARCH.

SEARCH ALL Elément-1 [AT END Ordre Impératif-1]
WHEN Condition-1 { Ordre Impératif-2 / NEXT SENTENCE }

En français :

EXPLORER TOUS LES Elément-1 [A LA FIN faire Ordre Impératif-1]
Si la Condition-1 est satisfaite faire { Ordre Impératif-2 / PHRASE SUIVANTE }

Elément-1 doit appartenir à une table c'est-à-dire être lié à une clause ØCCURS et surtout doit être associé à une clause

{ ASCENDING / DESCENDING } KEY

car la condition-1 doit obligatoirement faire intervenir un des arguments de classement (KEY) liés à Elément-1.

REMARQUE 1 : Si une clef intervient dans la condition-1, tout argument de classement qui lui est supérieur doit intervenir dans la condition-1.

Par exemple pour :

```
02  RESID ØCCURS 1000 ASCENDING KEY
                                    DEPT VILLE INDEXED BY IND.
    04 DEPT  PIC 99.
    04 VILLE PIC 999.
    04 RUE   PIC    X(30).
```

le fait de faire intervenir VILLE dans condition-1 oblige à faire intervenir DEPT argument de classement supérieur à VILLE.

On peut alors écrire WHEN DEPT = 62 AND VILLE = 100 GØ TØ SUITE.

REMARQUE 2 : Condition-1 ne pourra faire intervenir que des signes égal (=) et exclusivement des opérations logiques AND s'il s'agit d'une condition composée. Ce format 2 SEARCH ALL, destiné à l'exploration complète d'une table, utilise systématiquement le premier index associé à Elément-1 pour progresser dans la table mais il ne sera pas nécessaire de positionner au préalable ce premier index par l'ordre SET car COBOL s'en charge.

A noter qu'on peut aussi utiliser le format 1 pour explorer l'ensemble d'une table mais que les performances de traitement seront nettement moins bonnes.

Exemple d'utilisation du Format 2 : Recherche d'un numéro de compte analytique à partir d'une table de codes divisions et codes services.

```
A       B
01      TABLE.
        02  ENREG ØCCURS 1000 ASCENDING KEY
                              DIVISIØN SERVICE INDEXED BY IND.
        04  DIVISIØN PIC 99
        04  SERVICE  PIC  9(4).
        04  CØMPTE   PIC  9(4).
        04  LIBEL    PIC  X(20).

        SEARCH ALL ENREG AT END GØ TØ ERREUR
                WHEN DIVISIØN = DIV AND SERVICE = SERV
                        MØVE CØMPTE (IND) TØ NUCØMPTE.
```

Nous avons remarqué l'impossibilité d'explorer l'ensemble d'une table à plusieurs indices avec un seul ordre SEARCH mais en fait on peut y arriver à l'aide d'une boucle PERFØRM associée au verbe SEARCH.

Exemple : Nous rechercherons la plus haute valeur d'une table à trois indices.

```
A       B

77      I1  PIC  S9(9)  CØMP.
77      I2  PIC  S9(9)  CØMP.
77      RESERVE PIC S9(7)V99 CØMP.
01      PAGE ØCCURS 12 INDEXED BY IND1 IND11.
        02  LIGNE ØCCURS 20 INDEXED BY IND2 IND22.
            04  CØL ØCCURS 10 PIC S9(7)V99
                                  CØMP INDEXED BY IND3 IND33.

        PERFØRM CUMUL THRU FINCUM VARYING I1 FRØM 1
          BY 1 UNTIL I1 > 12 AFTER I2 FRØM 1 BY 1 UNTIL I2 > 20.
CUMUL.
        SET IND1 TØ I1          SET IND2 TØ I2
        SEARCH CØL AT END GØ TØ FINCUM
            WHEN CØL (IND1, IND2, IND3) > RESERVE MØVE
                 CØL (IND1, IND2, IND3) TØ RESERVE SET IND33
                   TØ IND3 SET IND11 TØ IND1 SET IND22 TØ IND2.
FINCUM. EXIT.
```

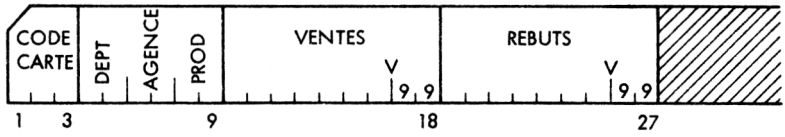
Nous avons utilisé les indices I1 et I2 pour le PERFØRM et à chaque boucle il faut transformer les occurences I1 et I2 en adresses d'index IND1 et IND2. Les index IND11, IND22 et IND33 servent à mémoriser l'adresse de la plus haute valeur.

REMARQUE : Rappelons que la dimension d'une table ne peut excéder à ce jour 32 767 octets.

EXERCICE N° 2

A partir de cartes comportant des quantités de produits vendus et rebutés, on désire cumuler toutes ces quantités et calculer le pourcentage de rebuts par produit dans chaque agence.

Les cartes lues ont le format suivant :



Il peut y avoir 50 départements, 20 agences et 10 produits.

Lire les cartes non classées pour charger une table, calculer les pourcentages puis les éditer sous la forme :

XX␣␣XX␣␣␣␣999,99␣␣␣␣999,99␣␣␣␣999,99 ---------- 999,99␣␣␣␣␣␣----␣␣

Dept Agence ←——————————————————→

dix pourcentages de produits

CHAPITRE X

LE TRI COBOL

PRINCIPES GÉNÉRAUX

Plutôt que de recourir aux EXIT de tri, le langage COBOL permet d'inclure les programmes généralisés de tri dans un programme COBOL.

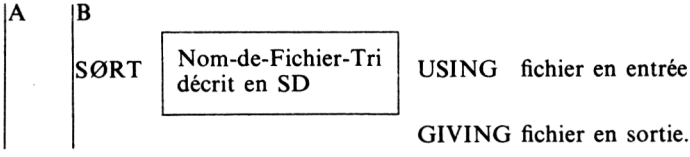
En fait, disons plus justement qu'on permet l'introduction de séquences de programmes COBOL, en début ou fin de tri, dans le but de modifier ou éliminer des enregistrements avant ou après tri.

Le programme avec tri est bâti différemment des programmes habituels (dans lesquels généralement on lit, on calcule, on écrit) car ici on ne lit ni écrit, simplement on trie (SØRT) et éventuellement on greffe une procédure en début de tri (INPUT PRØCEDURE) ou fin de tri (ØUTPUT PRØCEDURE).

Autre originalité des programmes de tri, l'apparition d'une description de fichier de tri SD en FILE SECTIØN au lieu des FD habituelles car il ne s'agira pas de décrire réellement un fichier mais les enregistrements à trier.

Les programmes de tri sont organisés comme suit :

Tri sans modification des enregistrements en entrée et sortie :



Il n'est pas utile d'ouvrir, lire, écrire ou fermer les fichiers entrée et sortie car le tri gère l'ensemble des fichiers.

Tri avec modification des enregistrements en entrée (INPUT PRØCEDURE) :

```
A    B
     SØRT  Nom-de-Fichier-Tri décrit en SD   INPUT PRØCEDURE Nom-de-Section
                                             GIVING fichier en sortie
Nom-de-Section SECTIØN.
                    ØPEN      fichier entrée
                    READ      fichier entrée
                    RELEASE   Enregistrement du fichier-tri décrit en SD.
```

Tri avec modification des enregistrements en sortie (ØUTPUT PRØCEDURE)

```
A    B
     SØRT  Nom-de-Fichier-Tri décrit en SD   USING   Fichier en entrée
                                             ØUTPUT PRØCEDURE
                                                    Nom-de-Section
Nom-de-Section SECTIØN.
                    ØPEN      Fichier en sortie
                    RETURN    Nom-de-Fichier-Tri décrit en SD   AT END Ordre Impératif
                    WRITE     Enregistrement du fichier en sortie.
```

Tri avec modification des enregistrements en entrée et en sortie :

```
A    B
     SØRT  Nom-de-Fichier-Tri décrit en SD   INPUT   PRØCEDURE
                                                    Nom-de-Section-1
                                             ØUTPUT PRØCEDURE
                                                    Nom-de-Section-2.
Nom-de-Section-1 SECTIØN.
                    READ      Fichier Entrée
                    RELEASE   Enregistrement du fichier-tri décrit en SD.
Nom-de-Section-2 SECTIØN.
                    RETURN    Nom-de-Fichier-Tri décrit en SD   AT END Ordre Impératif
                    WRITE     Enregistrement du fichier en sortie.
```

FORMAT GÉNÉRAL

Le format général d'un programme TRI COBOL est le suivant :

```
A     B
IDENTIFICATIØN DIVISIØN.
      {
ENVIRØNMENT DIVISIØN.
INPUT-ØUTPUT SECTIØN.
FILE-CØNTRØL.

      SELECT Nom-Fichier-1 en Entrée
                ASSIGN TØ Nom-Fichier-Système en entrée
                [RESERVE { Nombre-Entier } ALTERNATE AREA].
                         { NØ          }

      SELECT Nom-Fichier-2 en Sortie
                ASSIGN TØ Nom-Fichier-Système en sortie
                [RESERVE { Nombre Entier } ALTERNATE AREA].
                         { NØ          }

      SELECT Nom-Fichier-3 de Tri
                ASSIGN TØ [Nombre-Entier]Nom-Système-1
                          [Nom-Système-2] ....

I-Ø-CØNTRØL.
            [RERUN ØN Nom-de-Fichier-Système]

      [SAME { RECØRD } AREA FØR Nom-Fichier-i1
            { SØRT  }
                                  [Nom-Fichier-i2] ...]

DATA  DIVISIØN.
FILE  SECTIØN.

FD    Nom-Fichier-1 en Entrée
                      }
FD    Nom-Fichier-2 en Sortie
                      }
SD    Nom-Fichier-3 de Tri      [RECØRDING MØDE IS Mode]

        [RECØRD CØNTAINS [entier-1 TØ] entier-2 CHARACTERS]
        [DATA RECØRD IS Nom-d'Enreg^t-1 [Nom-d'Enreg^t-2] ....].

01    Nom-d'Enregistrement-1.
                      } Description de (ou des) l'enregistrement de tri.
```

```
A    B
PRØC EDURE DIVISIØN.
     SØRT Nom-de-Fichier-3 de Tri
                    ØN { DESCENDING } KEY [Argument 1] ....
                       { ASCENDING  }
                  [ØN { DESCENDING } KEY [Argument 2] .... ]
                       { ASCENDING  }
          { INPUT   PRØCEDURE IS Nom-Section-1                  }
          {                      [THRU Nom-Section-11]          }
          { USING   Nom-Fichier-1 en Entrée                     }
          { ØUTPUT  PRØCEDURE IS Nom-Section-2                  }
          {                      [THRU Nom-Section-22]          }
          { GIVING  Nom-Fichier-2 en Sortie                     }
[Nom-Section-1 SECTIØN.
          RELEASE Nom-Enregistrement-Fichier-3 [FRØM Nom-zone].
 Nom-Section-11 SECTIØN. EXIT.]
[Nom-Section-2 SECTIØN.
          RETURN Nom-Fichier-3 de Tri RECØRD [INTØ Nom-zone]
                                  AT END Ordre-Impératif.
 Nom-Section-22 SECTIØN. EXIT.]
```

LES INSTRUCTIONS COBOL

SELECT Nom-Fichier-1-en-Entrée ----
SELECT Nom-Fichier-2-en-Sortie -----

La clause SELECT, obligatoire pour tous les fichiers, l'est en particulier pour les fichiers en entrée et sortie du programme de tri pour faire la liaison entre les noms de fichiers du programme COBOL et ceux des cartes JCL de contrôle du système d'exploitation. Notons toutefois qu'il ne sera pas nécessaire d'écrire des clauses SELECT pour les fichiers intermédiaires de tri (SØRTWØRK).

```
SELECT Nom-Fichier-3 de Tri
     ASSIGN TØ [Nbre-Entier] Nom-Système-1 [Nom-Système-2] ....
```

La clause SELECT doit aussi être obligatoirement utilisée pour le fichier à trier bien que cela semble être en contradiction avec ce que nous venons d'écrire au sujet des fichiers intermédiaires de tri.

En fait, cette clause ne sert pratiquement qu'à identifier le fichier-Tri, donc le format des enregistrements à trier, pour le compilateur COBOL.

La clause ASSIGN est pourtant obligatoire et doit en principe indiquer le nombre et les noms des fichiers intermédiaires de tri ; mais en fait cette clause ne sert que de commentaire si elle est contredite par le contenu des cartes JCL de contrôle d'exploitation.

Nombre-Entier : désigne le nombre de fichiers intermédiaires (SØRT-WØRK) utilisés. Nom-Système-1, Nom-Système-2 ... désignent ces fichiers intermédiaires.

Exemple : SELECT TRAVAIL ASSIGN TØ UT-S-SØRTWK 01
UT-S-SØRTWK 02
UT-S-SØRTWK 03

[RERUN ØN Nom-de-Fichier-Système]

Cette clause, utilisée dans le paragraphe I-Ø-CØNTRØL permet de faire générer des points de reprise au cours du déroulement du tri.

Les intervalles de temps seront fonction du nombre d'enregistrements triés, et calculés directement par le programme de tri.

Nom-de-Fichier-Système doit naturellement désigner le fichier destiné à recevoir les informations des points de reprise.

[SAME { RECØRD / SØRT } AREA FØR Nom-Fichier-i1
[Nom-Fichier-i2] ....]

Lorsque RECØRD est utilisé, cette clause permet d'économiser de la place mémoire en utilisant une même zone buffer en mémoire centrale pour des fichiers utilisés lors de phases distinctes du programme.

L'option SØRT ne sert à rien car, le but du programme de tri étant d'optimiser les performances de tri, il ne saurait être question de laisser à quiconque la possibilité de le dégrader.

A	B
SD	Nom-Fichier-3 de Tri --------

Une description SD (SØRT DESCRIPTIØN) est obligatoire pour chaque fichier à trier et cette description suit les mêmes règles que les descriptions de fichiers FD (FILE DESCRIPTIØN) mises à part les clauses BLØCK CØNTAINS et LABEL RECØRD qui n'ont ici aucune utilité.

Nous ne reviendrons donc pas sur ces règles d'écriture des descriptions (1). Rappelons qu'en fait on ne décrit pas un fichier-Tri réellement mais des enregistrements qu'il va falloir trier et qui seront délivrés au programme de tri par une INPUT PRØCEDURE ou par le fichier-1 en entrée (option USING).

(1) Voir l'ouvrage précédent « LE COBOL A. N. S. » du même auteur (Eyrolles éditeur).

Les enregistrements de tri pourront être de format fixe F, Variable V ou Spanned S.

SØRT Nom-Fichier-3 de Tri ØN { DESCENDING / ASCENDING } KEY [Argument-1] .....

L'ordre SØRT, placé dans la PRØCEDURE DIVISIØN, désigne les enregistrements à trier, c'est-à-dire le fichier-3 décrit en Data Division par SD.

{ DESCENDING / ASCENDING } KEY désigne l' (ou les) argument de tri Argument 1, ...

Pour préciser la hiérarchie majeur, inter, mineur... des arguments de tri, on renouvellera autant que nécessaire la clause { DESCENDING / ASCENDING } KEY.

Par exemple : SØRT FICHTRI ØN ASCENDING MAJEUR DESCENDING INTER ASCENDING MINEUR.

DESCENDING signifie que le classement doit être décroissant donc du plus haut vers le plus bas alors que ASCENDING indique un classement croissant du bas vers le haut.

Restrictions : Les clefs doivent êtres communes aux divers types d'enregistrements (s'il en existe plusieurs) et placées à des adresses fixes par rapport au début des enregistrements.

Pour un fichier, il ne peut y avoir plus de 12 arguments de tri et la somme des longueurs de ces arguments ne peut excéder 256 octets ; enfin les clefs doivent être contenues dans les 4 092 premiers octets si la Longueur de l'enregistrement est supérieure à ce nombre.

USING Nom-Fichier-1 en Entrée

Lorsque USING est indiqué, tous les enregistrements du fichier-1 en entrée sont utilisés par le tri, c'est-à-dire que le programmeur renonce à éliminer des enregistrements avant tri ou plus simplement à effectuer des travaux quelconques sur les enregistrements en entrée.

Notons que, avec USING, la gestion du fichier-1 est entièrement prise en compte par le tri, donc le programmeur n'a pas à écrire d'ordre ØPEN fichier-1, ni READ fichier-1, ni CLØSE fichier-1.

GIVING Nom-Fichier-2 en Sortie.

L'option GIVING signifie que tous les enregistrements triés seront

directement livrés au fichier-2 en sortie ; donc le programmeur renonce à travailler sur les enregistrements triés ou à en éliminer au passage.

Comme pour USING, la gestion du fichier-2 est entièrement prise en charge par le tri, ce qui dispense d'écrire les ordres ØPEN, WRITE et CLØSE pour ce fichier-2.

INPUT PRØCEDURE IS Nom-Section-1 [THRU Nom-Section-11]

Cette clause est utilisée pour désigner une section-1 ou une suite de sections (de section-1 à Section-11) dans lesquelles le programmeur désire travailler sur les enregistrements en entrée avant de les délivrer au tri par l'ordre RELEASE.

Rappelons que l'ensemble du programme tri COBOL est sous la dépendance de l'ordre SØRT. La (ou les) section INPUT PRØCEDURE doit donc être écrite d'une manière totalement indépendante comme s'il n'y avait rien d'autre autour. On y lit le fichier-1, on travaille les enregistrements-1 puis on livre les enregistrements qu'il faut trier, les uns après les autres, par l'ordre RELEASE.

Il n'est pas nécessaire de programmer un GØ TØ ou un PERFØRM pour entrer dans cette section, le tri s'en charge seul au fur et à mesure qu'il faut obtenir des enregistrements à trier.

Dans la section INPUT, il est possible de programmer tous travaux désirés par le programmeur, exceptés :

— l'interdiction de placer un ordre SØRT dans une telle SECTIØN,
— l'interdiction de se brancher à l'extérieur des sections INPUT par GØ TØ ou PERFØRM,
— l'interdiction de se brancher dans une section INPUT autrement que par l'effet du verbe SØRT.

ØUTPUT PRØCEDURE IS Nom-Section-2 [THRU Nom-Section-22]

Section-2 et Section-22 désignent la (ou les) section dans laquelle le programmeur travaille sur les enregistrements en sortie de tri avant de les écrire sur le fichier-2 en Sortie.

La section ØUTPUT est une section indépendante : les enregistrements en sortie de tri sont fournis à la section ØUTPUT par un ordre RETURN qui constitue en quelque sorte l'entrée dans la section — ces enregistrements sont travaillés dans la section puis écrits sur le fichier-2 en sortie.

En fin de section ØUTPUT, le contrôle est automatiquement repassé au tri. De manière à éviter toute ambiguïté sur la fin de la section ØUTPUT, nous recommandons d'utiliser la clause THRU Nom-Section-22 et l'ordre EXIT. Par exemple :

```
A    B
          ØUTPUT PRØCEDURE DEBSØRTIE THRU FINSØRTIE.
DEBS ØRTIE SECTIØN.
FINS ØRTIE SECTIØN. EXIT.
```

Dans la section ØUTPUT il est possible de réaliser tous travaux désirés par le programmeur avec les mêmes restrictions que pour les sections INPUT :

— l'interdiction d'y placer un ordre SØRT,
— l'interdiction de sortir des sections ØUTPUT par GØ TØ ou PERFØRM,
— l'interdiction de se brancher dans une section ØUTPUT autrement que par l'ordre RETURN.

RELEASE Nom-Enregistrement-Fichier-3 [FRØM Nom-zone]

L'ordre RELEASE ne peut être utilisé que dans une section INPUT et est destiné à fournir au module de tri les enregistrements à trier sous le format Nom Enregistrement fichier 3, éventuellement à partir d'une zone grâce à l'option FRØM Nom-zone.

RETURN Nom-Fichier-3 de Tri
RECØRD [INTØ Nom-Zone] AT END Ordre-Impératif

L'ordre RETURN ne peut être placé que dans une section ØUTPUT et est destiné à recevoir du module de tri les enregistrements du fichier Nom-Fichier-3 de Tri qui vient d'être trié et de les délivrer éventuellement dans une zone particulière par l'option INTØ Nom-Zone. C'est en quelque sorte un ordre de lecture du fichier intermédiaire de tri.

AT END Ordre Impératif indique la procédure à suivre lorsque tous les enregistrements ont été traités.

Remarque importante : L'INPUT ou l'ØUTPUT PRØCEDURE doit être une séquence totalement indépendante aussi, par mesure de sécurité, le programmeur veillera à n'entrer qu'une seule fois dans l'INPUT ou l'ØUTPUT PRØCEDURE par le verbe SØRT et de même à ne sortir qu'une seule fois pour retourner au verbe SØRT (voir l'exemple).

LES COMPTEURS SPÉCIAUX DU TRI

Le tri COBOL utilise quatre compteurs spéciaux accessibles au programmeur et permettant, comme pour les utilitaires de tri, d'améliorer les performances de tri en fournissant :

— la taille du fichier à trier = SØRT – FILE – SIZE
— la taille de mémoire centrale utilisable = SØRT – CØRE – SIZE
— la taille la plus fréquente des enregistrements en variable = SØRT – MØDE – SIZE.

SØRT – FILE – SIZE : est une zone binaire de PICTURE S9(8) CØMP destinée à recevoir l'estimation du nombre d'enregistrements à trier afin de permettre au module de tri d'optimiser l'utilisation de ses ressources en conséquence.
Dans le cas où cette information n'est pas fournie, le module de tri utilise le maximum de ressources disponibles comme si le nombre d'enregistrements à trier devait être maximum.

SØRT—CØRE—SIZE : est aussi une zone binaire de PICTURE S9(8) CØMP destinée à recevoir la taille de mémoire centrale utilisable par le module de tri, si celle-ci est différente des spécifications normales, de manière à ce que la taille des monotonies soit modifiée en conséquence.

SØRT – MØDE – SIZE : est une zone binaire de PICTURE S9(5) CØMP réservée aux tris d'enregistrements de longueur variable.

Il est en effet préférable d'indiquer au module de tri la taille la plus fréquente des enregistrements à trier. Par exemple, pour un fichier historique pouvant avoir des enregistrements de longueur variable de 100 à 500 octets mais n'ayant pratiquement en début de carrière que des enregistrements de 150 octets, il conviendrait de charger la valeur 150 dans SØRT – MØDE – SIZE afin d'améliorer les performances de tri.

Les trois compteurs n'ont pas à être déclarés en DATA DIVISIØN mais doivent être chargés par un ordre MØVE avant le passage sur l'ordre SØRT pour être efficaces et en aucun cas il ne sera possible de les utiliser avec les verbes ACCEPT, DISPLAY ou EXHIBIT puisqu'il s'agit de quantités binaires. Le programmeur devra donc procéder en deux temps, par exemple :

```
ACCEPT LØNGUEUR – MØYENNE
MØVE LØNGUEUR – MØYENNE TØ SØRT – MØDE – SIZE.
```

Le quatrième compteur de tri SØRT – RETURN est une zone binaire de PICTURE S9(4) chargé d'une valeur de 0 à 16 en fin de travail par le tri pour en indiquer le succès (zéro) ou la faillite (seize). Ceci peut permettre d'éviter ou non la poursuite des programmes suivants du JØB par test du code condition.

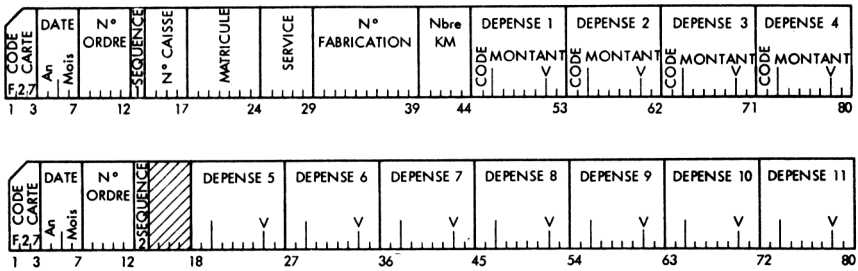
De toutes manières, SØRT – RETURN utilisé ou non, les commentaires sur le déroulement du tri seront toujours fournis par le système d'exploitation.

EXEMPLE DE PROGRAMME DE TRI

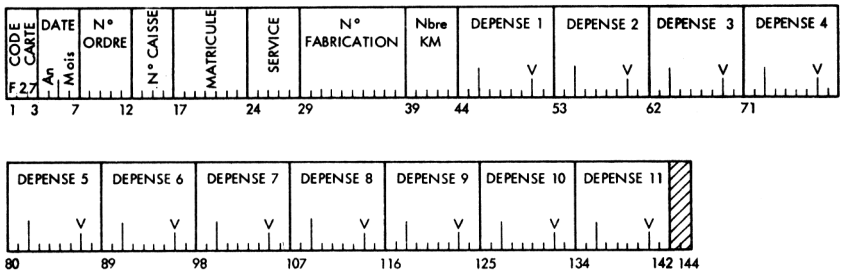
Il arrive parfois qu'on doive utiliser deux cartes pour contenir l'ensemble des données d'un enregistrement logique et on peut dans ce cas profiter du tri des enregistrements cartes pour reconstituer les enregistrements logiques.

Supposons des cartes frais de déplacements ; afin de permettre une comptabilité par nature, on enregistre le détail des frais grâce à des codes à deux positions suivis du montant correspondant. On suppose ne pas dépasser 11 codes frais pour une note de frais individuelle.

Le dessin des cartes en entrée est le suivant :



Les enregistrements logiques en sortie de tri ont le format suivant :




```
IDENTIFICATION  DIVISION.
PROGRAM-ID.  MISENFOR.
REMARKS.  PROGRAMME DE TRI AVEC MISE EN FORME DES
          ENREGISTREMENTS EN SORTIE.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
     SELECT  ENTREE       ASSIGN TO   UT-S-ENTREE.
     SELECT  SORTIE       ASSIGN TO   UT-S-SORTIE.
     SELECT  TRAVAIL      ASSIGN TO   UT-S-TRAVAIL.
DATA  DIVISION.
FILE  SECTION.
SD  TRAVAIL  DATA RECORD  ENR1 ENR2.
```

```
01  ENR1.
    02  TET1.
        04  CC1      PIC      X(3).
        04  DATE1    PIC      9(4).
        04  ORDRE1   PIC      X(5).
    02  SEQ1     PIC X.
    02  CORP1.
        04  CAISS1   PIC      9(4).
        04  MATR1    PIC      9(7).
        04  SERV1    PIC      X(5).
        04  OF1      PIC      X(10).
        04  KM1      PIC      9(5).
    02  VAL1     OCCURS 4     PIC  9(9).
01  ENR2.
    02  FILLER  PIC X(17).
    02  VAL2  OCCURS  7   PIC  9(9).
FD  ENTREE  LABEL RECORD STANDARD
            BLOCK CONTAINS 1600
            RECORDING MODE F
            DATA RECORD CARTE.
01  CARTE   PIC X(80).
FD  SORTIE  LABEL RECORD STANDARD
            BLOCK CONTAINS 1440
            RECORDING MODE F
            DATA RECORD BANDE.
01  BANDE.
    02  TET2     PIC   X(12).
    02  CORP2    PIC   X(31).
    02  VAL  OCCURS  11 PIC  9(9).
    02  FILLER  PIC   XX.
WORKING-STORAGE  SECTION.
77  I    PIC  S9(9)  COMP.
77  J    PIC  S9(9)  COMP.
PROCEDURE  DIVISION.
DEBUT.
    OPEN OUTPUT  SORTIE.
    MOVE SPACE TO BANDE.
TRI.
    SORT TRAVAIL ASCENDING KEY DATE1 ORDRE1 SEQ1  USING ENTREE
                 OUTPUT PROCEDURE  BATIBAND THRU FINBATI.
    STOP RUN.
BATIBAND  SECTION.
BATA.
    RETURN  TRAVAIL AT END  GO TO FINTRA.
    IF SEQ1 = '1'  GO TO CARTUN.
    IF TET1 NOT = TET2  WRITE BANDE  MOVE SPACE TO BANDE
        MOVE TET1 TO TET2.
    PERFORM METVAL VARYING I FROM 1 BY 1 UNTIL  I > 7
    WRITE BANDE MOVE SPACE TO BANDE
    GO TO BATA.
METVAL.
    COMPUTE  J = I + 4
    MOVE  VAL2 (I) TO VAL (J).
CARTUN.
    IF TET2 = SPACE  GO TO CARTE1.
    WRITE BANDE.
CARTE1.
    MOVE TET1  TO TET2
    MOVE CORP1 TO CORP2
    MOVE VAL1  (1)  TO VAL (1)
    MOVE VAL1  (2)  TO VAL (2)
    MOVE VAL1  (3)  TO VAL (3)
    MOVE VAL1  (4)  TO VAL (4).
    GO TO BATA.
FINBATI.   EXIT.
FINTRA.
    IF TET2 NOT = SPACE WRITE BANDE.
    CLOSE SORTIE.
```

Le verbe MERGE (fusion de fichiers).

Il est possible de fusionner des fichiers, déjà triés sur des arguments identiques, en utilisant le verbe MERGE.

La programmation est identique à celle du tri SØRT, à ceci près qu'il n'est pas admis de modification des enregistrements en entrée (INPUT PRØCEDURE). Le format d'écriture est donc le suivant :

```
MERGE Nom-de-fichier-de-tri
        ØN { DESCENDING } KEY Argument-1 ----
           { ASCENDING  }
      [ØN { DESCENDING } KEY Argument-2 ----]
           { ASCENDING  }
      USING Nom-de-fichier-1  Nom-de-fichier-2 [Nom-de-fichier-3]---
            en entrée         en entrée          en entrée
      { GIVING Nom-de-fichier-en-Sortie
      { ØUTPUT PRØCEDURE IS Nom-de-Section-1
                                  [THRU Nom-de-Section-2]
```

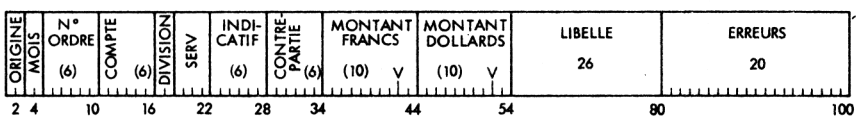
EXERCICE N° 3

Supposons un traitement de comptabilité mensuelle. Nous venons de réaliser le contrôle des écritures comptables classées sur le numéro de compte pour améliorer les temps de traitement. Nous avons donc des enregistrements logiques longs de 100 octets et dont les 20 derniers octets représentent les erreurs : 1 si erreur et blanc sinon. Nous désirons maintenant trier sur origine, mois, numéro d'ordre, éliminer en sortie les enregistrements ayant des codes erreurs de 81 à 100 et éditer les erreurs. Les enregistrements valides sont conservés mais réduits à 80 octets.

Les enregistrements en entrée ont le format suivant :

Arguments de tri croissants : Date, Numéro d'ordre, Séquence.

ORIGINE	MOIS	N° ORDRE (6)	COMPTE (6)	DIVISION	SERV	INDICATIF (6)	CONTRE-PARTIE (6)	MONTANT FRANCS (10) V	MONTANT DOLLARDS (10) V	LIBELLE 26	ERREURS 20
2	4	10	16		22	28	34	44	54	80	100



Les enregistrements en sortie sont semblables avec les codes erreurs en moins.

L'état d'erreur a le format suivant :

```
R    MOIS    ORDRE    COMPTE    DIV.    SERV    INDIC.    C. PART    MONTANT FR.    MONTANT DOL
XbbbbXXbbbbXXXXXXbbbbXXXXXXbbbbXXbbbbXXXXbbbbXXXXXXbbbbXXXXXXbbbbXXXXXXXX, XXbbbbXXXXXXXX, XXbbbb

RREURSbbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbbXXbb ------
```

CHAPITRE XI

LE REPORT WRITER ÉDITEUR COBOL

PRINCIPES GÉNÉRAUX

Le REPORT WRITER est un module généralisé permettant de générer automatiquement des états en déchargeant le programmeur des problèmes de ruptures, comptages de lignes, édition d'en-tête, cumuls intermédiaires et généraux, etc...

En édition, l'unité homogène de base est la ligne d'imprimante mais l'idée de base du REPORT WRITER est de rassembler les lignes en groupes d'édition.

Premier principe : Un ensemble ordonné de lignes éditées successivement et dépendant d'un conditionnement unique constitue un groupe d'édition.

Second principe : A un groupe d'édition succède obligatoirement un groupe d'édition unique dépendant d'un conditionnement.

Il résulte de ces principes que nous ne saurions trop recommander au programmeur, avant d'entreprendre la rédaction d'un programme d'édition, de commencer par analyser le problème pour dégager la liste des groupes d'édition et leurs conditionnements.

Pour un état, le programme REPORT WRITER doit comporter :

— Une déclaration de fichier en clause SELECT d'ENVIRØNMENT DIVISIØN ainsi qu'en FD de FILE SECTIØN (mais sans description d'enregistrement).

— Une description d'état en REPORT SECTIØN :

— lignes en-tête,
— lignes détail,
— lignes bas de page.

— Les verbes de génération d'état en PRØCEDURE DIVISIØN. INITIATE, GENERATE, TERMINATE.

— Éventuellement, des DECLARATIVES pour les traitements préalables à l'édition des lignes.
-----SECTIØN. USE BEFØRE REPØRTING.

FORMAT GÉNÉRAL D'UN PROGRAMME

Le format général d'un programme REPØRT WRITER est le suivant :

```
A       B
 IDENTIFICATIØN DIVISIØN.
 ENVIRØNMENT DIVISIØN.
 INPUT-ØUTPUT SECTIØN.
 FILE-CØNTRØL.
      SELECT Nom-Fichier-Etat ASSIGN TØ Nom-de-Fichier-Système.
 DATA  DIVISIØN.
  FILE SECTIØN.
FD     Nom-Fichier-Etat  LABEL RECØRD --------
                         [BLØCK CØNTAINS ------]
                         [RECØRD CØNTAINS ----]
                         [RECØRDING MØDE ----]

                          REPØRT IS Nom-Etat.
WØRKING-STØRAGE SECTIØN.
 REPØRT SECTIØN.
RD     Nom-Etat          [WITH CØDE --------]
                         [CØNTRØL IS-------]
                         [PAGE LIMIT IS -----].

01     [Nom-de-Ligne]
                         { Description de Groupe d'Edition.
 PRØCEDURE DIVISIØN.
[DECLARATIVES.                                              ]
[------SECTIØN. USE BEFØRE REPØRTING -----.                 ]
 END DECLARATIVES.

       INITIATE --------
       GENERATE -----
       TERMINATE ----
      STØP RUN.
```

ENVIRØNMENT DIVISIØN.

```
B
SELECT Nom-Fichier-Etat ASSIGN TØ Nom-de-Fichier-Système.
```

La clause SELECT est obligatoire pour le fichier édition comme pour tout autre fichier. Remarquons que rien ne s'oppose à ce que le fichier soit édité simultanément sur plusieurs supports, ce qui imposera autant de clauses SELECT que de supports pour le même état.

Exemple : Si on désire éditer un état sur bande magnétique pour envoi de la bande dans un autre centre informatique, il est prudent d'éditer simultanément sur bande et sur papier, d'où deux clauses SELECT.

FILE SECTIØN.

```
A       B
FD      Nom-Fichier-Etat
          LABEL RECØRD IS { ØMITTED
                          { STANDARD
                          { Nom-Donnée-1, Nom-Donnée-2 }
          BLØCK CØNTAINS Entier-1 [TØ Entier-2] [{ CHARACTERS }]
                                                [{ RECØRDS    }]
          [RECØRD CØNTAINS Entier-1 [TØ Entier-2]]
          RECØRDING MØDE IS Mode
          REPØRT IS Nom-Etat-1, [Nom-Etat-2]-----
```

Les clauses LABEL, BLØCK, RECØRD et RECØRDING ne présentent pas de particularités par rapport aux fichiers séquentiels et standards.

La clause REPØRT par contre remplace la clause DATA RECØRD et est destinée à lier les descriptions d'état Nom-Etat-1, Nom-Etat-2, ... au fichier Nom-Fichier-Etat. Toutefois, ici nous n'aurons pas les descriptions d'état à la suite en FILE SECTIØN mais dans une section spéciale, la REPØRT SECTIØN.

REMARQUE : Si on désire éditer l'état sur deux supports, une bande magnétique et un état papier, il faudra deux clauses SELECT et deux clauses FD pour une seule REPØRT SECTIØN.

```
|A      |B
|       |
|       |SELECT  EDITBAND  ASSIGN TØ  UT-S-EDITBAND.
|       |SELECT  EDITPAPI  ASSIGN TØ UT-S-EDITPAPI.
|DATA   | DIVISIØN.
| FILE  | SECTIØN.
|FD     |EDITBAND RECØRDING MØDE F
|       |         LABEL STANDARD
|       |         REPØRT ETAT.
|FD     |EDITPAPI RECØRDING MØDE F
|       |         LABEL STANDARD
|       |         REPØRT ETAT.
```

REPØRT SECTIØN.

La REPØRT SECTIØN se présente sous une forme analogue à celle de la FILE SECTIØN.

La description des paramètres généraux en RD (Report Description), puis la description des différents groupes d'édition constituant l'état.

```
|A      |B
|REPØ   |RT SECTIØN.
|RD     |Nom-Etat
|       |          [WITH CØDE Nom-Mnemonic]
|       |          [CØNTRØL IS { FINAL                          }]
|       |          [           { Identifier-1 [Identifier-2] --- }]
|       |          [           { FINAL Identifier-1 [Identifier-2] --- }]
|       |          [PAGE LIMIT IS Entier-1 LINE]
|       |          [HEADING Entier-2]
|       |          [FIRST DETAIL Entier-3]
|       |          [LAST DETAIL Entier-4]
|       |          [FØØTING Entier-5].
```

[WITH CØDE Nom-Mnemonic]

Cette clause est essentiellement utilisée dans le cas d'édition sur bande magnétique d'états imbriqués, généralement dans le but d'une future édition Spool. Chaque ligne d'état sur bande est alors précédée, devant le caractère de saut, d'un caractère alphanumérique permettant de distinguer chaque état.

Le Nom-Mnemonic correspond à un des noms du paragraphe SPECIAL-NAMES.

Par exemple, si on désire créer deux états imbriqués dont les enregistrements seront distingués respectivement par les codes A et B, on écrit :

```
A       B

 ENVI   RØNMENT DIVISIØN.
CØNF    IGURATIØN SECTIØN.
 SPEC   IAL-NAMES.
                    "A" IS ETA.
                    "B" IS ETB.
 INPU   T-ØUTPUT SECTIØN.
  FILE  -CØNTRØL.
          SELECT EDITBAND ASSIGN TØ UT-S-EDITBAND.

DATA      DIVISIØN.
 FILE     SECTIØN.
FD        EDITBAND  RECØRDING MØDE F
                    RECØRD CØNTAINS 134
                    LABEL STANDARD
                    REPØRT ETATA ETATB.
 REPØ   RT SECTIØN.
RD      ETATA CØDE ETA
                  ≀
RD      ETATB CØDE ETB
                  ≀
```

$$\left[\underline{\text{CØNTRØL}}\ \text{IS}\ \left\{\begin{array}{l}\underline{\text{FINAL}}\\ \text{Identifier-1 [Identifier-2]}\ \text{----------}\\ \underline{\text{FINAL}}\ \text{Identifier-1 [Identifier-2]}\text{-----}\end{array}\right\}\right]$$

« CØNTRØL » correspond aux groupes de lignes éditées spécialement en cas de ruptures dans l'état, c'est-à-dire que les groupes « CØNTRØL » résulteront en général de tests de zones de données, à chaque ligne détail, afin de détecter les ruptures éventuelles devant générer des lignes de totalisations partielles ou générales correspondant aux groupes d'édition. Les zones de tests de « CØNTRØL » doivent être énoncées dans l'ordre d'importance décroissante, majeures vers mineures.

Le plus haut niveau de « CØNTRØL » est le niveau FINAL de fin d'état et le seul qui ne consiste pas à tester une zone de donnée.

A chaque ligne détail, COBOL évalue les différents tests « CØNTRØL » et lorsqu'une rupture est décelée pour l'un d'entre eux, il y a naturellement génération de la cascade des niveaux de rupture qui lui sont inférieurs.

A chaque niveau de rupture ou conditionnement correspond un groupe d'édition CØNTRØL, la correspondance étant donnée lors de la description des groupes.

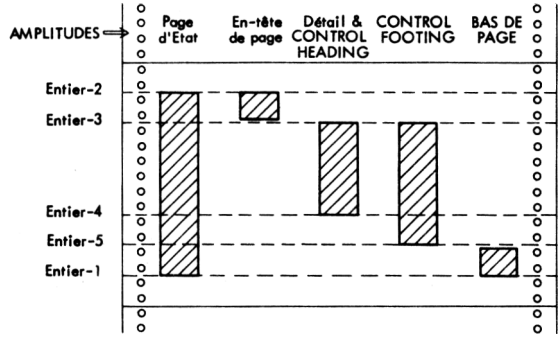
Par exemple, pour CØNTRØL IS PAYS DEPT CANTØN VILLE, une rupture détectée sur DEPT déclenchera les contrôles inférieurs CANTØN VILLE d'où la génération des groupes d'édition de totalisation associés à VILLE puis CANTØN puis DEPT avant un groupe d'édition en-tête de DEPT.

[PAGE LIMIT IS Entier-1 LINE]
[HEADING Entier-2]
[FIRST DETAIL Entier-3]
[LAST DETAIL Entier-4]
[FØØTING Entier-5]

Ces différentes clauses servent à délimiter la position des lignes dans la page.

Entier-1 : Désigne le nombre maximum de lignes par page de l'état.
Entier-2 : Désigne le numéro de la première ligne éditée en haut de page de l'état (généralement il s'agit d'une ligne titre).
Entier-3 : Désigne le numéro de la première ligne DETAIL qu'il est possible d'éditer et le numéro de ligne à partir duquel seront effectués les tests de CØNTRØL HEADING (en-tête) ou FØØTING (bas de page).
Entier-4 : Indique le numéro de l'ultime ligne DETAIL qu'il est possible d'éditer dans une page et l'ultime numéro de ligne de test pour le CØNTRØL HEADING.
Entier-5 : Désigne le dernier numéro de ligne où sera effectué le test de CØNTRØL FØØTING ainsi que le numéro de ligne après lequel seront éditées les lignes de bas de page.

On peut alors déduire de ces nombres l'amplitude des différentes zones de la page d'édition.



Amplitude de l'État : de Entier-2 à Entier-1.
En-tête de page : de Entier-2 à Entier-3 moins une ligne.
Lignes détail : de Entier-3 à Entier-4.
Totalisations Bas de page : de Entier-5 plus une ligne à Entier-1.
CØNTRØL HEADING : de Entier-3 à Entier-4 avec les lignes détail.
CØNTROL FØØTING : de Entier-3 à Entier-5.

Ces différentes clauses et entier-i ne sont pas obligatoires et dépendent de l'état considéré :

— Si Entier-1 est omis, le programmeur prend en charge totalement le comptage des lignes.

— Si Entier-2 est omis, le haut de page est assimilé à la ligne numéro 1.

— Si Entier-3 est omis, Entier-3 est assimilé à Entier-2 (ou a défaut à la ligne numéro 1).

— Si Entier-4 est omis, Entier-4 est assimilé à Entier-5, et réciproquement.

— Si Entier-5 est omis, Entier-5 est assimilé à Entier-4.

— Si Entier-4 et Entier-5 sont omis, ils sont alors assimilés à Entier-1.

LES DESCRIPTIONS DE GROUPES D'ÉDITIONS

Décrire un groupe d'édition consiste à :

— Indiquer les conditionnements de ce groupe d'édition.

— Décrire le contenu des lignes d'édition.

— Indiquer les totalisations à réaliser.

— Indiquer les procédures préalables (DECLARATIVES) qu'il convient de réaliser avant l'édition de groupes.

Les caractéristiques du groupe d'édition sont fournies au niveau 01 en marge A puis la description de chaque ligne du groupe est faite avec des niveaux de 02 à 49 ; par exemple :

```
A       B
01      Niveau Groupe
        02  Niveau ligne
            04  Niveau contenu de la ligne.
```

Lorsqu'un groupe d'édition ne comporte qu'une seule ligne, on pourra ne pas donner de nom de ligne donc se contenter du niveau groupe et du niveau contenu de ligne.

La répartition des clauses COBOL dans les niveaux groupe, ligne ou détail est très théorique car, à la limite, on peut avoir un groupe comportant une seule ligne composée elle-même d'une seule information détail ; l'ensemble des clauses peut alors être rassemblé au niveau groupe.

Description au Niveau groupe d'Edition :

Le format général de description au Niveau groupe d'Edition est le suivant :

```
A       B
01      Nom-de-Groupe  [LINE  NUMBER IS ----]
                       [NEXT  GROUP  IS------]
                        TYPE  IS ------------.
```

Description au Niveau ligne :

Le format général de description au Niveau ligne est le suivant :

```
A       B
        Nbre-Niveau [Nom-de-Ligne] [LINE NUMBER IS ----].
```

Description au Niveau contenu de ligne :

Le format général de description du contenu de ligne est le suivant :

```
A       B
        Nbre-Niveau [Nom-Elément] [CØLUMN NUMBER IS ----]
                                  [GRØUP INDICATE]
                                  [JUSTIFIED ----]
                                   PICTURE------
                                  [RESET ØN ----]
                                  [BLANK WHEN ZERØ]
                                  [{SØURCE}
                                   {SUM   } -----].
                                   {VALUE }
```

Voyons en détail chacune de ces clauses.

```
A       B
            LINE NUMBER IS { Nombre-1       }
                           { PLUS Nombre-2  }
                           { NEXT PAGE      }
```

Cette clause est obligatoire pour toute ligne d'état, sauf pour la première ligne d'un groupe d'édition si la clause a été spécifiée au niveau groupe.

LINE NUMBER indique la position dans la page où doit être écrite la ligne (où la première ligne du groupe d'édition). On peut utiliser :

— Nombre-1 pour désigner le numéro de ligne dans la page.

— Nombre-2 pour désigner la position relative, c'est-à-dire le nombre de lignes, par rapport à la précédente.

— NEXT PAGE pour indiquer que la ligne doit être écrite à la page suivante. Il ne peut donc s'agir ici que de la première ligne d'un groupe, ce qui réduit l'usage de cette option au niveau groupe d'édition ou à la première ligne d'un groupe.

```
A       B
            NEXT GRØUP IS { Nombre-1       }
                          { PLUS Nombre-2  }
                          { NEXT PAGE      }
```

La clause NEXT GRØUP est à utiliser lorsqu'on désire imposer la position relative du groupe d'édition suivant.

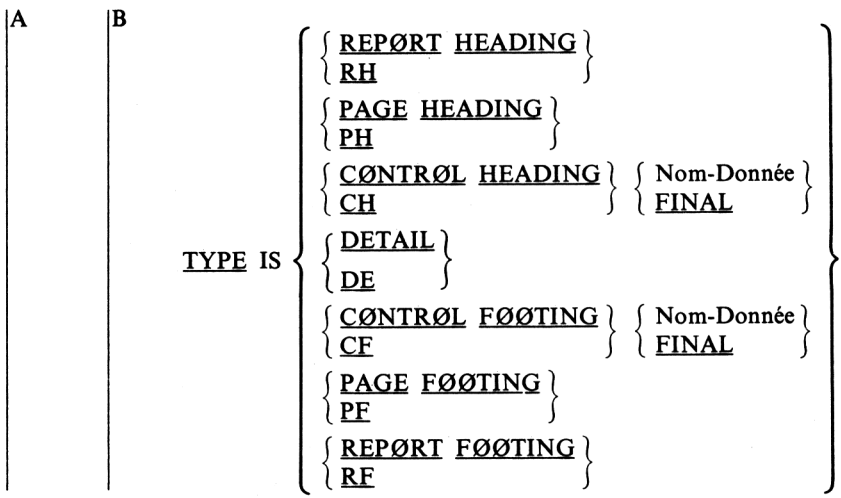
Mais en fait, on ne pourra utiliser cette clause que dans le groupe REPØRT HEADING, donc uniquement en tête d'état car la clause ne sera effective que si le test de « control » ayant entraîné l'édition du groupe est bien directement lié à ce groupe et non à un groupe de niveau supérieur ayant entraîné des éditions en cascade. Par exemple, si l'édition provient de la rupture sur MOIS, la clause NEXT GRØUP écrite pour le niveau JOUR n'aura pas d'effet.

Les trois options ont la même signification que dans la clause LINE :

— Nombre-1 désigne un numéro de ligne dans la page.

— Nombre-2 désigne le nombre de lignes dont il faut progresser après exécution de la dernière ligne du groupe en cours d'édition.

— NEXT PAGE signifie que le prochain groupe devra être édité à la page suivante.



La clause TYPE IS définit le type de groupe d'édition, c'est-à-dire le moment dans le cours du programme où ce groupe doit être édité de manière à ce que le REPORT WRITER puisse réaliser automatiquement la génération des lignes en temps opportun.

{ REPØRT HEADING / RH } désigne l'en-tête d'état qui sera édité une seule fois dès le premier ordre GENERATE de création de ligne.

{ PAGE HEADING / PH } désigne l'en-tête de page qui sera édité systématiquement au début de chaque page de l'état.

{ CØNTRØL HEADING / CH } le groupe d'édition CØNTRØL HEADING est bâti sous le groupe en-tête de page lorsqu'il y a un CØNTRØL ou rupture sur le Nom-Donnée ou en fin d'état (FINAL) suivant l'option utilisée.

En conséquence, si une rupture intervient sur Nom-Donnée, on passe à la page suivante pour éditer l'en-tête de page puis le groupe CØNTRØL HEADING.

Rappelons que Nom-Donnée ou FINAL doivent avoir été désignés dans la clause CØNTRØL de la REPØRT SECTIØN.

{ DETAIL / DE } Le groupe DETAIL est le groupe qui dirige pratiquement toute l'édition car c'est en fonction de l'ordre GENERATE ligne-Détail que le REPØRT WRITER génère les autres groupes d'édition.

En général, ce groupe se borne à une seule ligne mais rien ne s'oppose à ce qu'il y en ait plusieurs ou plus simplement que la ligne détail soit accouplée à une ligne d'astérisques par exemple.

{ CØNTRØL FØØTING / CF } Ce groupe d'édition est le pendant du CØNTRØL HEADING vu plus haut mais édité en bas de page (avant le groupe de bas de page) en cas de rupture sur Nom-Donnée ou en fin d'état (FINAL).

En conséquence, si une rupture intervient sur Nom-Donnée, on édite le groupe CØNTRØL FØØTING puis le bas de page (PAGE FØØTING), à la suite de quoi on passe à la page suivante.

Rappelons ici encore que Nom-Donnée ou FINAL doivent avoir été désignés dans la clause CØNTRØL de la REPØRT SECTIØN.

{ PAGE FØØTING / PF } le groupe de bas de page sera édité systématiquement pour chaque page de l'état si cette clause est spécifiée.

{ REPØRT FØØTING / RF } le groupe de fin d'état sera édité une seule fois lorsque toutes les lignes détail et control intermédiaires auront été éditées. Ce groupe REPØRT FØØTING de même que les CØNTRØL FØØTING FINAL seront générés à l'initiative de l'ordre TERMINATE.

```
|A      |B
        CØLUMN NUMBER IS Entier-1
```

Avec le REPØRT WRITER, il n'est pas nécessaire de décrire la totalité de la ligne y compris les blancs, mais uniquement les zones utiles.

La clause CØLUMN NUMBER permet de désigner la position d'une zone élémentaire dans la ligne d'état en indiquant le numéro (entier-1) de l'octet de gauche de la zone.

En principe les zones élémentaires d'une ligne doivent être décrites en partant de la gauche vers la droite, mais le COBOL IBM est indifférent à l'ordre.

Exemple : Pour décrire une ligne comportant uniquement le titre « ETAT DES VENTES », on écrira : 04 CØLUMN 40 PICTURE X(17) VALUE "ETAT DES VENTES".

Ce qui générera 39 caractères blancs, 17 caractères contenant « ETAT DES VENTES » et le reste de la ligne en caractères blancs.

A	B
	GRØUP INDICATE

La clause GRØUP INDICATE est destinée à éviter la répétition du contenu d'un indicatif lorsque celui-ci ne varie pas. L'indicatif n' est alors édité qu'une seule fois à la première occurence de chaque valeur, c'est-à-dire après détection d'une rupture sur la zone. L'indicatif est aussi édité automatiquement sur la première ligne DETAIL de chaque page, même si aucune rupture n'a été détectée, de manière à ce qu'on ait pas à rechercher l'information sur des pages antérieures.

Par exemple, dans une liste de primes individuelles, on ne répétera pas le matricule ni le nom de l'intéressé à chacune des primes, ce qui donnera :

23456	DUBELLAC MICHEL	2240	ANCIENNETE	150,32
		2513	PØSTE	65,45
		2918	TECHNICITE	100,00
		2957	PRØDUCTIVITE	316,18
		9999	TØTAL	631,95
25316	EVENTIER JACQUES	2240	ANCIENNETE	180,70
		2918	TECHNICITE	100,00
		9999	TØTAL	280,70

A	B
	JUSTIFIED-----
	PICTURE-----
	BLANK WHEN ZERØ

Ces trois clauses ne présentent pas de particularités par rapport aux clauses COBOL habituelles (1).

(1) Voir l'ouvrage précédent « LE COBOL A. N. S. » du même auteur (Eyrolles éditeur).

```
A       B
        SØURCE IS { Nom-Donnée }
                  { TALLY      }
```

La clause SØURCE est destinée à remplacer un ordre MØVE sur la zone d'édition. Lorsque la ligne correspondant à la zone élémentaire concernée est éditée, il y a automatiquement chargement du contenu de la zone à partir de la « SØURCE ».

Exemple :

```
A       B
        02  LI5  LINE 5.
            04  CØLUMN 30 PICTURE ZZ9V,99 SØURCE QUANTITE.
```

A chaque édition de la ligne LI5, le REPØRT WRITER effectue un MØVE de QUANTITE sur la zone élémentaire qui commence en colonne 30 et de PICTURE ZZ9V,99. Nom-Donnée est une zone quelconque de DATA DIVISIØN pouvant être remplacée par TALLY : compteur spécial de 5 chiffres sans signe, normalement réservé au verbe EXAMINE et mis à la disposition du programmeur pour tout usage.

```
A       B
        SUM { Identifier-1 } [ Identifier-2 ] ..... [UPØN Nom-Donnée]
            { TALLY        } [ TALLY        ]
```

La clause SUM permet de faire réaliser des totalisations intermédiaires par le REPØRT WRITER dans des compteurs internes.

Ces totalisations ne seront normalement éditées que dans les groupes d'édition CØNTRØL FØØTING lorsque les ruptures CØNTRØL auront été détectées.

Exemple :

```
01      TYPE CØNTRØL FØØTING MØIS.
        02  LI6  LINE PLUS 4.
            04  CØLUMN 40 PIC X(15) VALUE "TOTAL DU MØIS".
            04  CØLUMN 60 PIC ZZZBZZ9V,99 SUM MØNTANT.
```

La ligne LI6 ne sera éditée que lorsque seront détectées des ruptures sur MOIS donc en fin de 01, 02, 03, etc. et sur cette ligne apparaîtra TØTAL DU MOIS suivi de la somme des montants du mois.

Le cumul repart à zéro après chaque rupture sauf si l'ordre RESET est spécifié.

S'il est besoin de faire référence à la somme dans le cours du programme

en PRØCEDURE DIVISIØN, il suffit de donner un nom à la zone élémentaire d'édition, par exemple :

04 SØMME CØLUMN 60 PIC 9(9) SUM MØNTANT.

La taille du compteur interne de somme est définie implicitement par la PICTURE d'édition associée à SUM, il est donc prudent de définir cette PICTURE assez largement pour éviter des troncations intempestives.

[UPØN Nom-Donnée] est utilisée lorsqu'on désire un cumul de plusieurs sommes associées à des clauses SUM. Nom-Donnée doit alors désigner une zone élémentaire d'édition DETAIL.

```
|A      |B
        |   VALUE IS litteral
```

La clause VALUE du REPØRT WRITER ne présente pas de particularités par rapport à la clause COBOL habituelle.

```
|A      |B
        |   RESET ØN { Identifier-1 }
        |            { FINAL        }
```

La clause RESET ne peut être spécifiée que pour des zones élémentaires déclarées CØNTRØL en REPØRT SECTIØN.

Cette clause RESET est utilisée pour réaliser des cumuls de totaux intermédiaires. En effet, les totaux intermédiaires, édités en groupe CØNTRØL FØØTING sont normalement remis à zéro après leur édition ; si la clause RESET est spécifiée on obtient alors une addition de la somme jusqu'à ce qu'il y ait une rupture sur Identifier-1.

Identifier-1 est donc obligatoirement un « CØNTRØL » de plus haut niveau que la zone élémentaire du RESET et doit avoir été spécifié dans la clause CØNTRØL et en REPØRT SECTIØN.

Lorsque FINAL est utilisé, il y a totalisation d'un bout à l'autre du programme sans aucune remise à zéro intermédiaire.

Exemple : 04 TØTALGEN CØLUMN 60 PIC ZZZBZZ9V,99 SUM TØTAL RESET FINAL.

OPTIONS PERMISES OU NON SUIVANT LES GROUPES D'ÉDITION

— REPØRT HEADING :

— LINE NEXT PAGE ne peut être utilisé car ce groupe est obligatoirement le premier de l'état.

— Si on désire que le groupe soit seul sur une page, on doit écrire NEXT GRØUP IS NEXT PAGE.

— PAGE HEADING, PAGE FØØTING :

— LINE NEXT PAGE ne doit pas être utilisé car ce groupe est obligatoirement édité en haut de page.

— La clause NEXT GRØUP ne peut être utilisée car ce groupe est dépendant des autres et non pas directif.

— CØNTRØL HEADING, DETAIL ET CØNTRØL FØØTING :

— LINE NEXT PAGE peut être utilisé.

— La clause NEXT GRØUP peut être utilisée pour passer à la page suivante (NEXT PAGE) ou imposer une ligne d'édition au prochain groupe à venir.

— REPØRT FØØTING :

— LINE NEXT PAGE peut être utilisé si on désire que le groupe soit édité sur une nouvelle page.

— NEXT GRØUP ne peut être utilisé puisqu'il n'y a normalement plus de groupe à éditer après cette fin d'état.

LES COMPTEURS SPÉCIAUX DE L'ÉDITEUR COBOL

Il existe trois compteurs spéciaux PAGE-CØUNTER (compteur de page), LINE-CØUNTER (compteur de ligne) et PRINT-SWITCH (aiguillage d'édition) gérés par l'éditeur COBOL et qu'il est inutile de décrire en DATA DIVISIØN.

PAGE-CØUNTER désigne des compteurs numériques d'usage CØMPUTATIØNAL-3 (condensé) sur trois octets, PICTURE S9(5).

PAGE-CØUNTER est initialisé à un au départ puis est automatiquement incrémenté de un à chaque fois qu'un CØNTRØL de page est détecté. Le programmeur peut donc utiliser ce compteur pour éditer le numéro de page de l'état dans le groupe PAGE HEADING d'en-tête de page (ou ailleurs). Il peut y avoir autant de compteurs PAGE-CØUNTER qu'il a été décrit de Noms-d'Etats en REPØRT SECTIØN mais, dans ce cas, le programmeur ne doit pas oublier de qualifier PAGE-CØUNTER par le Nom-d'Etat correspondant.

LINE-CØUNTER désigne des compteurs numériques d'usage CØMPUTATIØNAL-3 (condensé) sur trois octets PICTURE S9(5).

LINE-CØUNTER est initialisé à zéro au départ puis est automatiquement incrémenté du nombre de lignes éditées ou dont on progresse

dans la page. Lorsque le bas de page est atteint, c'est-à-dire lorsque la valeur entier-1 de la clause PAGE LIMIT est atteinte, le compteur de lignes est automatiquement remis à zéro et un CØNTRØL de page est généré.

Le programmeur peut utiliser ce compteur le cas échéant pour éditer le numéro de ligne ; on peut aussi en changer la valeur mais on risque alors de perturber sérieusement l'édition.

Il peut y avoir autant de LINE CØUNTER qu'il existe de Noms-d'Etats décrits en REPØRT SECTIØN mais dans ce cas il faut qualifier les compteurs lorsqu'il est fait référence à l'un d'eux.

PRINT-SWITCH désigne des compteurs numériques ne pouvant prendre que les valeurs zéro ou un et ne pouvant être utilisés qu'en SECTIØN DECLARATIVE, USE BEFØRE REPØRTING Nom-de-Groupe d'Edition.

L'éditeur COBOL génère autant de compteurs PRINT-SWITCH qu'il existe de groupes d'éditions. Le but de ces aiguillages est de supprimer l'édition du groupe correspondant lorsque la valeur du PRINT-SWITCH est à 1. Au départ les PRINT-SWITCH sont initialisés à zéro, ce qui leur confère une neutralité totale. Si le programmeur désire éliminer l'édition d'un groupe, il écrira dans la Section Déclarative correspondante MØVE 1 TØ PRINT-SWITCH. Il faudra donc autant de Sections Déclaratives et de MØVE 1 TØ PRINT-SWITCH qu'on désire éliminer de groupes en édition.

Naturellement, lorsque l'édition d'un groupe est annulée, l'incrémentation de LINE-CØUNTER est annulée de même que l'effet de la clause NEXT GRØUP.

LA PRØCÉDURE DIVISIØN

Le traitement proprement dit doit commencer par l'ouverture des fichiers et en particulier par l'ouverture du fichier Nom-Fichier-Etat devant recueillir les enregistrements lignes de l'état.

Puis l'édition sera initiée par un ordre INITIATE, les lignes seront générées par l'ordre GENERATE, l'édition sera terminée par l'ordre TERMINATE.

Enfin les fichiers devront être fermés.

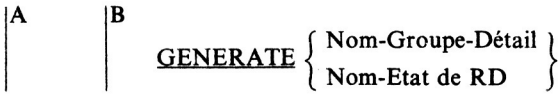
A	B
	INITIATE Nom-Etat-1 [Nom-Etat-2] ....

L'instruction INITIATE est destinée à initialiser tous les compteurs internes de l'éditeur COBOL avant de démarrer la production d'un état.

En particulier, rappelons que LINE-CØUNTER est mis à zéro, PAGE-CØUNTER est initialisé à 1 et PRINT-SWITCH à zéro.

Cette instruction INITIATE ne remplace pas, et doit intervenir après, l'ordre ØPEN d'ouverture des fichiers. Elle ne produit pas non plus l'édition du groupe REPØRT HEADING (en-tête d'état) produit par le premier ordre GENERATE. Nom-Etat-1, Nom-Etat-2 ... doivent avoir été définis au préalable en REPØRT SECTIØN.

Notons enfin qu'il est possible d'écrire un second INITIATE pour un état, à condition qu'il ait été précédé par une instruction TERMINATE sur le même état.



L'instruction GENERATE est destinée à produire un état. Les effets de GENERATE sont différents suivant qu'on spécifie Nom-Groupe-Détail ou Nom-Etat de RD. Nom-Groupe-Détail est utilisé dans le cas général où on édite au moins un groupe détail pour l'état considéré.

Les effets de l'ordre GENERATE sont les suivants :

— Approvisionne le contenu des différentes zones des lignes d'édition.

— Produit les groupes d'édition de l'état REPØRT HEADING, PAGE HEADING, PAGE FØØTING et DETAIL.

— Réalise tous les cumuls déclarés par SUM et gère les compteurs spéciaux internes.

— Gère les ruptures CØNTRØL HEADING et FØØTING, à l'exclusion des CØNTRØL FINAL.

— Le groupe REPØRT FØØTING sera produit par l'ordre TERMINATE.

Nom-Etat de RD est utilisé dans le cas d'états récapitulatifs, c'est-à-dire lorsqu'on ne désire pas éditer de lignes détails mais seulement des cumuls. Tous les cumuls sont alors effectués comme pour un ordre GENERATE ligne détail, à l'exception de la production des éventuelles lignes détail associées au Nom-Etat de RD.

Notons qu'on peut à la fois générer normalement des lignes détail et produire un état récapitulatif.

Même s'il existe plusieurs groupes DETAIL d'édition, donc plusieurs ordres GENERATE, les tests de rupture de CØNTRØL ne sont effectués qu'une seule fois pour un état en début d'itération. Puis les cumuls des différentes SUM sont effectués dans l'ordre de leur apparition dans la REPØRT SECTIØN.

Qu'il s'agisse d'un GENERATE « détail » ou d'un GENERATE « récapitulatif », les opérations sont effectuées dans l'ordre suivant :

— Positionnement et test de LINE-CØUNTER et PAGE-CØUNTER pour produire éventuellement les groupes de bas de page et haut de page (PAGE HEADING et PAGE FØØTING).

— Test des ruptures de CØNTRØL pour produire éventuellement les groupes CØNTRØL FØØTING et CØNTRØL HEADING.

— Cumul des compteurs désignés par les clauses SUM.

— Exécution des routines spéciales programmées en sections DECLARATIVES.

— Mouvement des données en zone d'édition.

```
|A      |B
|       |   TERMINATE,Nom-Etat-1 [Nom-Etat-2] ...
```

L'instruction TERMINATE est utilisée pour achever l'édition d'un état. Elle génère les groupes CØNTRØL FØØTING FINAL et CØNTRØL HEADING FINAL et le groupe de fin d'état REPØRT FØØTING.

Cette instruction ne remplace et doit précéder l'ordre CLØSE de fermeture des fichiers.

Comme pour l'instruction INITIATE, on ne programme un second TERMINATE pour un état qu'à condition qu'il soit précédé d'un second INITIATE.

LES SECTIONS DECLARATIVES
USE BEFØRE REPØRTING

```
|A      |B
|DECL   |ARATIVES.
|
| Nom-  |Section SECTIØN.
|       |        USE BEFØRE REPØRTING Nom-Groupe d'Edition.
| END   | DECLARATIVES.
```

Une section déclarative USE BEFØRE REPØRTING est utilisée pour exécuter des traitements juste avant l'édition du groupe désigné dans la clause.

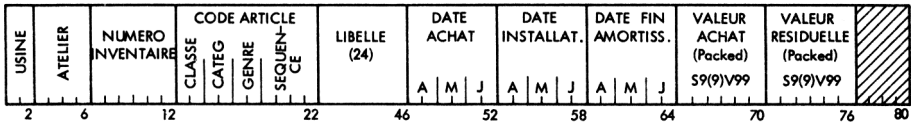
Une section déclarative peut être associée à n'importe quel groupe d'édition sauf à un groupe DETAIL. Cette restriction est normale car l'édition d'un groupe DETAIL est commandée par l'instruction GENERATE avant laquelle il est possible de programmer toute séquence désirée. Par contre, l'édition des autres groupes est dirigée par COBOL, ce qui impose de se débrancher à une séquence indépendante en l'occurence une section déclarative.

Rappelons qu'il n'est en principe pas permis de sortir d'une section Déclarative et faire référence à des Noms paragraphes extérieurs à cette section, mais que le CØBØL IBM le permet. Réciproquement, on ne peut entrer directement dans une section déclarative à partir de la Division Procédure, mais le CØBØL IBM le permet (1).

Rappelons aussi que c'est dans les sections déclaratives qu'on peut éliminer ou restaurer l'édition d'un groupe grâce au compteur spécial PRINT-SWITCH.

EXEMPLE DE PROGRAMME REPORT WRITER

A partir d'un fichier séquentiel dont les enregistrements logiques ont le format ci-dessous, on désire éditer l'inventaire du matériel d'une société possédant plusieurs usines et plusieurs ateliers par usine.



La première page de l'état ne comporte que deux lignes :

ligne 5, à partir de la colonne 37 « SØCIETE DES FØRGES DE l'ESCAUT »

ligne 8, à partir de la colonne 31 « INVENTAIRE GENERAL DU MATERIEL AU XX/XX/XX » (la date à éditer est celle du jour de traitement en ordinateur).

Le format des pages courantes est indiqué ci-après.

Pour chaque atelier, chaque usine et pour l'ensemble de la société on édite la valeur totale du matériel.

L'intitulé des usines sera obtenu par une table de 20 éléments, chargée à partir de cartes ACCEPT comprenant uniquement le code usine (en colonnes 1 et 2) suivi du nom de l'usine (20 caractères). Une carte avec code usine 99 ferme ce fichier.

La première opération à réaliser est de déterminer les groupes d'édition composant cet état :

Premier groupe : En-tête d'état ou REPØRT HEADING, c'est la première page de l'état comportant les deux lignes :

SØCIETE DES FØRGES -----
INVENTAIRE GENERAL-----

(1) Voir l'ouvrage précédent « LE COBOL A. N. S. » du même auteur.

```
|1        10|        20|        30|        40|        50|        60|        70|        80|        90|        100|
1          USINE DE    XXXXXXXXXXXXXXXXXXXX                    INVENTAIRE AU    XX/XX/XX          PAGE   XXXX  Li1
2
3
4          ATELIER     XXXX                                                                                     Li4
5
6
7  ***************************************************************************************************************  Li7
8  * NUMERO *      CODE      *                          *   DATE    *   DATE    *    FIN    *                  *  Li8
9  * INVENT *     ARTICLE    *          LIBELLE         *   ACHAT   *   INSTAL   *   AMORT   *      VALEUR      *  Li9
10 ***************************************************************************************************************  Li10
11 *        *                *                          *           *           *          *                  *  Li11
12 * XXXXXX * XX XX XX XXXX * XXXXXXXXXXXXXXXXXXXXXXXX * XX XX XX * XX XX XX * XX XX XX * XXX XXX XXX ,XX *  Li12
13 *        *                *                          *           *           *          *                  *  Li13
14 * XXXXXX * XX XX XX XXXX * XXXXXXXXXXXXXXXXXXXXXXXX * XX XX XX * XX XX XX * XX XX XX * XXX XXX XXX ,XX *
15 *        *                *                          *           *           *          *                  *
   *        *                *                          *           *           *          *                  *

   ***************************************************************************************************************  LiA
   *                                                                                      *                  *  LiB
   *                                                  TOTAL      ATELIER                  *  XXX XXX XXX ,XX *  LiC
   *                                                                                      *                  *  LiD
   ***************************************************************************************************************  LiE

   ***************************************************************************************************************  LiA
   *                                                                                      *                  *  LiB
   *                                                  TOTAL      USINE                    *  XXX XXX XXX ,XX *  LiC
   *                                                                                      *                  *  LiD
   ***************************************************************************************************************  LiE

   ***************************************************************************************************************  LiA
   *                                                                                      *                  *  LiB
   *                                      TOTAL GENERAL SOCIETE                           *  XXX XXX XXX ,XX *  LiC
   *                                                                                      *                  *  LiD
   ***************************************************************************************************************  LiE
```

```
IDENTIFICATION  DIVISION.
PROGRAM-ID. INVENTER.
REMARKS. EXEMPLE  D 'UTILISATION  DU  REPORT WRITER.
ENVIRONMENT  DIVISION.
INPUT-OUTPUT  SECTION.
FILE-CONTROL.
    SELECT  FILECT  ASSIGN TO   UT-S-FILECT.
    SELECT  FIETAT  ASSIGN TO   UT-S-FIETAT.
DATA  DIVISION.
FILE  SECTION.
FD  FILECT  RECORDING  MODE F
            LABEL RECORD STANDARD
            DATA  RECORD ENTREE.
01  ENTREE.
    02  USINE        PIC      99.
    02  ATELIER      PIC      9(4).
    02  INVENT       PIC      9(6).
    02  ARTIC        PIC      9(10).
    02  LIBEL        PIC      X(24).
    02  DATACH       PIC      9(6).
    02  DATINST      PIC      9(6).
    02  DATFIN       PIC      9(6).
    02  VALACH       PIC      S9(9)V99 COMP-3.
    02  VALRES       PIC      S9(9)V99 COMP-3.
    02  FILLER       PIC      X(4).
FD  FIETAT  RECORDING MODE F
            LABEL STANDARD
            REPORT ETAT.
WORKING-STORAGE  SECTION.
01  CARTE.
    02  CARTUS  PIC   99.
    02  CARLIB  PIC   X(20).
    02  FILLER  PIC   X(58).
01  TABUSINE.
    02  ETAB  OCCURS  20.
        04  CODUS   PIC   99.
        04  LIBUS   PIC   X(20).
REPORT  SECTION.
RD  ETAT    CONTROL FINAL USINE ATELIER
            PAGE LIMIT 60 LINE
            HEADING  1
            FIRST DETAIL 12
            LAST DETAIL 54
            FOOTING  56.
01  TETETAT  TYPE REPORT HEADING    NEXT GROUP NEXT PAGE.
    02  LINE  5.
        04  COLUMN 37 PIC X(34) VALUE
              'SOCIETE  DES  FORGES  DE  L ESCAUT'.
    02  LINE  PLUS 3.
        04  COLUMN 31 PIC X(37) VALUE
              'INVENTAIRE  GENERAL  DU  MATERIEL  AU'.
        04  COLUMN 70 PIC X(8)  SOURCE  CURRENT-DATE.
01  TETPAGE TYPE PAGE HEADING.
    02  LI1 LINE 1.
        04  COLUMN  11  PIC X(9)    VALUE   'USINE  DE'.
        04  COLUMN  22  PIC X(20)   SOURCE  LIBUS (USINE).
        04  COLUMN  63  PIC X(14)   VALUE   'INVENTAIRE  AU'.
```

```
      04  COLUMN  79  PIC X(8)     SOURCE  CURRENT-DATE.
      04  COLUMN  96  PIC X(4)     VALUE   'PAGE'.
      04  COLUMN  102 PIC ZZZ9     SOURCE  PAGE-COUNTER.
   02 LI4 LINE 4.
      04  COLUMN  11  PIC X(7)     VALUE   'ATELIER'.
      04  COLUMN  21  PIC 9(4)     SOURCE  ATELIER.
   02 LI7 LINE 7.
      04  COLUMN  2   PIC X(104)   VALUE  ALL '*'.
   02 LI8 LINE 8.
      04  COLUMN  2   PIC X(26) VALUE '* NUMERO *      CODE'.
      04  COLUMN  27  PIC X     VALUE '*'.
      04  COLUMN  54  PIC X(23) VALUE '*   DATE    *   DATE'.
      04  COLUMN  76  PIC X(12) VALUE '*     FIN    *'.
      04  COLUMN  105 PIC X     VALUE '*'.
   02 LI9 LINE 9.
      04  COLUMN  2   PIC X(26) VALUE '* INVENT *     ARTICLE'.
      04  COLUMN  27  PIC X     VALUE '*'.
      04  COLUMN  37  PIC X(7)  VALUE 'LIBELLE'.
      04  COLUMN  54  PIC X(23) VALUE '*  ACHAT    *  INSTAL'.
      04  COLUMN  76  PIC X(22) VALUE '*    AMORT   *     VALEUR'.
      04  COLUMN  105 PIC X     VALUE '*'.
   02 LI10 LINE 10.
      04  COLUMN  2   PIC X(104)   VALUE  ALL '*'.
   02 LI11  LINE 11.
      04  COLUMN  2   PIC X   VALUE   '*'.
      04  COLUMN  11  PIC X   VALUE   '*'.
      04  COLUMN  27  PIC X   VALUE   '*'.
      04  COLUMN  54  PIC X   VALUE   '*'.
      04  COLUMN  65  PIC X   VALUE   '*'.
      04  COLUMN  76  PIC X   VALUE   '*'.
      04  COLUMN  87  PIC X   VALUE   '*'.
      04  COLUMN  105 PIC X   VALUE   '*'.
01 LIDET  TYPE DETAIL.
   02 LI12  LINE PLUS 1.
      04  COLUMN  2   PIC X   VALUE  '*'.
      04  COLUMN  4   PIC 9(6)  SOURCE INVENT.
      04  COLUMN  11  PIC X   VALUE  '*'.
      04  COLUMN  13  PIC 99B99B99B9999 SOURCE ARTIC.
      04  COLUMN  27  PIC X   VALUE  '*'.
      04  COLUMN  29  PIC X(24) SOURCE LIBEL.
      04  COLUMN  54  PIC X   VALUE  '*'.
      04  COLUMN  56  PIC 99B99B99  SOURCE DATACH.
      04  COLUMN  65  PIC X   VALUE  '*'.
      04  COLUMN  67  PIC 99B99B99  SOURCE DATINST.
      04  COLUMN  76  PIC X   VALUE  '*'.
      04  COLUMN  78  PIC 99B99B99  SOURCE DATFIN.
      04  COLUMN  87  PIC X   VALUE  '*'.
      04  COLUMN  89  PIC ZZZBZZZBZZ9BV,99 SOURCE VALRES.
      04  COLUMN  105 PIC X   VALUE  '*'.
   02 LI13  LINE PLUS 1.
      04  COLUMN  2   PIC X   VALUE   '*'.
      04  COLUMN  11  PIC X   VALUE   '*'.
      04  COLUMN  27  PIC X   VALUE   '*'.
      04  COLUMN  54  PIC X   VALUE   '*'.
      04  COLUMN  65  PIC X   VALUE   '*'.
      04  COLUMN  76  PIC X   VALUE   '*'.
      04  COLUMN  87  PIC X   VALUE   '*'.
```

```
        04  COLUMN  105 PIC X    VALUE    '*'.
01  LINATELIER TYPE CONTROL FOOTING ATELIER
                NEXT GROUP NEXT PAGE.
    02  LINE PLUS 1 COLUMN 2 PIC X(104) VALUE ALL '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X         VALUE '*'.
        04  COLUMN  54   PIC X(17)   VALUE 'TOTAL      ATELIER'.
        04  COLUMN  87   PIC X       VALUE  '*'.
        04  COLUMN  89   PIC ZZZBZZZBZZ9BV,99 SUM VALRES.
        04  COLUMN  105 PIC X         VALUE  '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE PLUS 1 COLUMN 2 PIC X(104) VALUE ALL '*'.
01  LINUSINE TYPE CONTROL FOOTING USINE   NEXT GROUP NEXT PAGE.
    02  LINE PLUS 1 COLUMN 2 PIC X(104) VALUE ALL '*'.
    02  LINE  PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE  PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  54   PIC X(15) VALUE 'TOTAL      USINE'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  89   PIC ZZZBZZZBZZ9BV,99  SUM  VALRES.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE  PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE PLUS 1 COLUMN 2  PIC X(104) VALUE  ALL '*'.
01  LINTOTAL TYPE CONTROL FOOTING FINAL  LINE NEXT PAGE.
    02  LINE PLUS 1  COLUMN 2  PIC X(104) VALUE  ALL  '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X    VALUE   '*'.
        04  COLUMN  40   PIC X(27) VALUE 'TOTAL     GENERAL'.
        04  COLUMN  87   PIC X    VALUE   '*'.
        04  COLUMN  89   PIC ZZZBZZZBZZ9BV,99  SUM  VALRES.
        04  COLUMN  105 PIC X    VALUE   '*'.
    02  LINE PLUS 1.
        04  COLUMN  2    PIC X    VALUE    '*'.
        04  COLUMN  87   PIC X    VALUE    '*'.
        04  COLUMN  105 PIC X    VALUE    '*'.
    02  LINE PLUS 1 COLUMN 2 PIC X(104) VALUE  ALL '*'.
PROCEDURE  DIVISION.
DEBUT.
    OPEN  INPUT FILECT  OUTPUT FIETAT.
LEG1.
```

```
    ACCEPT CARTE     IF CARTUS = 99  GO TO LEC2.
    MOVE CARLIB TO LIBUS (CARTUS)    GO TO LEC1.
LEC2.
    READ FILECT AT END DISPLAY  'FICHIER VIDE' GO TO FINTRA.
    INITIATE  ETAT.
ECRITURE.
    GENERATE  LIDET.
    READ FILECT  AT END GO TO FINDETA.
    GO  TO ECRITURE.
FINDETA.
    TERMINATE  ETAT.
FINTRA.
    CLOSE FILECT  FIETAT     STOP RUN.
```

Second groupe : Tête de page ou PAGE HEADING. Il est composé des lignes nommées LI1, LI4, LI7, LI8, LI9, LI10, LI11.

Troisième groupe : Lignes DETAIL. Groupe Composé des lignes LI12 et LI13.

Quatrième groupe : Total atelier ou CØNTRØL FØØTING ATELIER. Groupe composé des lignes LIA, LIB, LIC, LID, LIE.

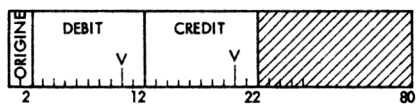
Cinquième groupe : Total usine ou CØNTRØL FØØTING USINE. Groupe composé des lignes LIA, LIB, LIC, LID, LIE à l'intitulé près.

Sixième groupe : Total société ou CØNTRØL FØØTING TØTAL. Identique aux groupes 4 et 5 à l'intitulé près.

EXERCICE N° 4

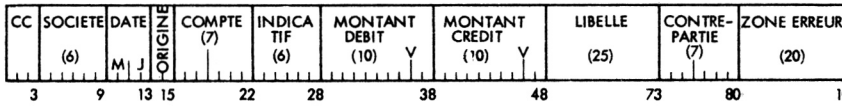
Supposons une application de comptabilité générale comportant plusieurs étapes et traitant des enregistrements logiques de 80 octets. Au cours des premières étapes ont été détectées des erreurs numérotées 01, 02, 03, ... mais comme on désire éditer toutes les erreurs sur une seule liste, on a allongé les enregistrements jusque 100 octets pour une zone erreur dont on a positionné les octets à 1 en cas d'erreur, à blanc sinon. Le rang de l'octet indique le type d'erreur octet 1 = erreur 01, octet 2 = erreur 02, etc...

On demande, outre l'édition de la liste des enregistrements erronés avec leurs codes erreurs, de signaler les origines (une origine = un certain nombre d'enregistrements) pour lesquelles le total des débits et le total des crédits ne sont pas égaux aux montants introduits par cartes paramètres de format ci-dessous.



Ces cartes sont lues par ACCEPT et la fermeture est obtenue par une carte d'origine 99.

Le fichier entré est classé par origine et a le format ci-dessous.



La liste des erreurs doit être éditée suivant le format ci-après.

— liste des codes erreurs avec leurs libellés sur la première page de l'état.

— liste des enregistrements erronés sur les pages suivantes. Pour chaque enregistrement erroné on édite trois lignes : ligne intitulée de zones, ligne enregistrement, lignes codes erreurs.

— En cas d'inégalité sur les cumuls d'une origine, on édite les trois lignes correspondant à ORIGINE XX ERRONEE DEBIT -----

Enfin, les enregistrements valides sont écrits sur un fichier sortie (toujours 100 octets de longs). Les enregistrements erronés ne doivent pas participer aux cumuls débits et crédits. Lorsqu'il y a égalité sur les débits et égalité sur les crédits, on ne doit rien éditer de particulier.

```
|1        10|        20|        30|        40|        50|        60|        70|        80|        90|        100|

1                          LISTE DES ANOMALIES DE COMPTABILITÉ                                    PAGE 1

5   01  ERREUR CODE CARTE
6   02  ERREUR CODE SOCIETE
7   03  ERREUR DATE
8   04  ORIGINE  INVALIDE
9   05  COMPTE  INVALIDE
10  06  INDICATIF INVALIDE
11  07  MONTANT DEBIT ERRONE
12  08  MONTANT CREDIT ERRONE
13  09  CONTRE-PARTIE INVALIDE
14  10  INCOMPATIBILITE SOCIETE*ORIGINE*COMPTE
15  11  INCOMPATIBILITE SOCIETE*ORIGINE*CONTRE-PARTIE
```

```
|1        10|        20|        30|        40|        50|        60|        70|        80|        90|        100|

1                          LISTE DES ANOMALIES DE COMPTABILITÉ                                    PAGE ZZZ9
2
3   CC      SOCIETE    DATE    OR     COMPTE     INDIC.      DEBIT        CREDIT              LIBELLE              C. PART.
4   XXX     XXXXXX     XXXX    XX     XXXXXXX    XXXXXX    XXXXXXXXXX   XXXXXXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX    XXXXXXX
5   ERREURS     01  02   03    04       06
6
7
8   CC      SOCIETE    DATE    OR     COMPTE     INDIC.      DEBIT        CREDIT              LIBELLE              C. PART
9   XXX     XXXXXX     XXXX    XX     XXXXXXX    XXXXXX    XXXXXXXXXX   XXXXXXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX    XXXXXXX
10  ERREURS     01        03        05  06  07
11
12                                          THEORIQUE     XX XXX XXX, XX                THEORIQUE  XX XXX XXX, XX
13  ORIGINE     XX  ERRONEE        DEBIT    **************************        CREDIT    ***********************
14                                          CALCULE       XX XXX XXX, XX                CALCULE    XX XXX XXX, XX
```

CHAPITRE XII

LA SEGMENTATION DES PROGRAMMES

Un obstacle assez fréquemment rencontré par le programmeur est la place mémoire centrale allouée inférieure à celle qui serait nécessaire pour exécuter les gros programmes dont il est le plus fier. On incrimine la machine, le langage, les contraintes d'exploitation absurdes, etc... mais l'usage montre que plus on augmente les ressources en taille mémoire plus les programmes ont tendance à devenir volumineux et dépasser les normes prescrites.

On commence donc par rogner sur les zones de travail puis sur les zones buffer d'entrée-Sortie. Mais la dégradation des performances devenant tellement importante et les modifications de maintenance ne faisant qu'aggraver le cas, on finit par se résoudre à envisager la segmentation des programmes, plus connue dans le jargon informatique sous le nom d'OVERLAY.

EN QUOI CONSISTE LA SEGMENTATION

La segmentation consiste à découper un programme en parties homogènes de telle manière que les parties les plus souvent exécutées soient de préférence en permanence dans la mémoire centrale de l'ordinateur et que les parties les moins souvent utilisées soient stoquées en attente sur mémoire à accès sélectif (disques, tambours) et introduites en mémoire centrale uniquement lorsqu'elles sont indispensables. Par exemple, dans le calcul du salaire du personnel de l'armée, il est assez peu fréquent d'avoir recours au vétérinaire conventionné pour soigner la mascotte du régiment ; ce qui fait que la partie spéciale de programme correspondante peut être stoquée sur disque en attendant les utilisations éventuelles.

Dans un programme COBOL, on ne pourra segmenter que la partie PRØCEDURE ; si bien que, au minimum, toute la DATA DIVISIØN devra être constamment présente en mémoire centrale. Notons aussi au passage que les sections DECLARATIVES devront être présentes en permanence dans la mémoire centrale.

REMARQUE : Avec les techniques de mémoire virtuelle, la segmentation jusqu'ici limitée à l'OVERLAY devient pratiquement indispensable pour optimiser l'utilisation de l'ordinateur.

ON DISTINGUE TROIS SORTES DE SEGMENTS DE PROCÉDURE

— *Les segments Résidents :* Ce sont les segments de procédure constamment présents en mémoire centrale. On dit parfois le « tronc du programme ».

— *Les segments Semi-Résidents :* Ce sont les segments en principe implantés en mémoire centrale mais qui sont évacués temporairement sur disque (ou sauvegardés) lorsqu'il faut dégager de la place mémoire pour des segments non-résidents.

— *Les segments Non-Résidents :* Ce sont les segments normalement stoqués sur disque et appelés en mémoire centrale que très occasionnellement.

REMARQUE IMPORTANTE : La différence essentielle entre Semi-Résidents et Non-Résidents est que :

— les Semi-Résidents sont évacués sur disque puis rappelés dans l'état où ils se trouvaient au moment de leur évacuation.

— Les Non-Résidents sont toujours appelés en mémoire centrale dans leur état initial car ils ne sont jamais évacués mais simplement écrasés en mémoire centrale par le segment successeur (c'est le segment de bibliothèque qui est appelé à chaque fois).

Classification des segments :

La première tâche pour l'analyste est de découper le programme en modules indépendants et de leur affecter un coefficient de priorité, les uns par rapport aux autres, pour résider en mémoire centrale. Un *numéro de priorité* est donc affecté à chacun des modules, de zéro à 99 dans l'ordre d'importance décroissante.

Priorité zéro : Pour les modules indispensables en permanence en mémoire centrale.

Priorités de 1 à 49 : Pour les modules très fréquemment utilisés.
Priorités de 50 à 99 : Pour les modules assez rarement sollicités.

On donnera, autant que possible, à des modules indépendants mais utilisés conjointement des numéros de priorité identiques.
On dit que plusieurs modules ayant un même nombre de priorité constituent un *segment*.

RÉDACTION DU PROGRAMME

Chacun des modules indépendants analysés à la page précédente sera programmé sous forme de SECTION dans la PROCEDURE du programme COBOL ; à chaque Nom de section étant accolé le numéro de priorité qui lui est affecté.

```
|A      |B
 Nom-   |Section SECTIØN [Numéro de Priorité].
```

Par exemple :

```
|A      |B
|CALC   |ULBRUT  SECTIØN 20.
```

définit une section appelée CALCULBRUT à laquelle un numéro de priorité égal à 20 a été affecté.

Le numéro de priorité doit être un nombre de zéro à 99 comme nous l'avons vu plus haut et par défaut le compilateur COBOL affecte le numéro zéro, c'est-à-dire que COBOL en fait une section résidente inamovible comme pour la Data Division et les sections Déclaratives.

On obtient alors un programme ayant sensiblement l'allure générale ci-contre.

— La section SECUNE de priorité zéro est indispensable et inamovible.

— Les sections SECDEUX et SECTRØIS seront très fréquemment utilisées et constituent un segment car le même numéro de priorité 20 suppose que ce sont des sections utilisées conjointement.

```
|A      |B
 IDEN   |TIFICATION DIVISIØN.
        §
 ENVI   |RØNMENT DIVISIØN.
        §
 DATA   | DIVISIØN.
        §
 PRØC   |EDURE DIVISIØN.
 SECU   |NE SECTIØN 0.
        §
 SECD   |EUX SECTIØN 20.
        §
 SECT   |ROIS SECTIØN 20.
        §
 SEQU   |ATRE SECTIØN 60.
        §
 SECS   |IX SECTIØN 40.
        §
 SECS   |EPT SECTIØN 60.
        §
 SECH   |UIT SECTIØN 50.
```

— La section SECSIX est un peu moins utilisée que SECDEUX et SECTRØIS.

— Les sections SEQUATRE et SECSEPT sont peu utilisées mais constituent un segment car elles sont utilisées conjointement (numéro de priorité identique 60).

REMARQUE : Rappelons que le fait de découper la PROCEDURE DIVISION en sections n'entraîne aucune contrainte supplémentaire.

LA RÉPARTITION DES SEGMENTS

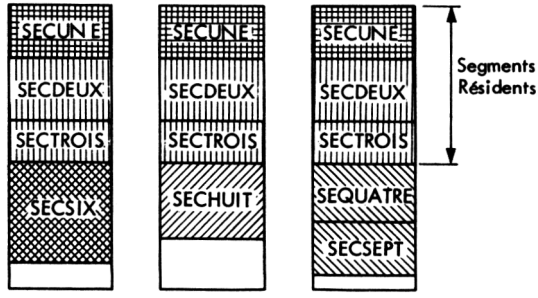
La répartition des segments en Résidents, Semi-Résidents et Non-Résidents est enfin réalisée en fonction de la place occupée par les différentes sections et de la taille de mémoire centrale disponible.

Par exemple, si comme l'indique le schéma ci-contre, il est possible de faire tenir en mémoire :

— SECUNE, SECDEUX, SECTRØIS et SECSIX au plus, ou :

— SECUNE, SECDEUX, SECTRØIS et SECHUIT au plus, ou :

— SECUNE, SECDEUX, SECTRØIS, SEQUATRE et SECSEPT au plus.



On déclarera

Segments Résidents : SECUNE, SECDEUX, SECTRØIS
Segment Semi-Résident : SECSIX
Segments Non-Résidents : SECHUIT, SEQUATRE, SECSEPT.

Cette déclaration est réalisée dans la DIVISIØN ENVIRØNMENT, paragraphe ØBJECT-CØMPUTER, par la clause SEGMENT-LIMIT qui a le format suivant :

A	B
ENVI	RØNMENT DIVISIØN.
CØNF	IGURATIØN SECTIØN.
ØBJE	CT-CØMPUTER. IBM-360 [– modèle] [SEGMENT-LIMIT IS Numéro-de-Priorité].

Par exemple, dans notre cas écrire : SEGMENT-LIMIT 30 détermine que les segments de nombre priorité < 30 (SECUNE, SECDEUX, SECTRØIS) sont résidents.

Les segments de Nombre priorité ⩾ 30 et < 50 (SECSIX) sont Semi-Résidents.

Les segments de nombre priorité ⩾ 50 (SECHUIT, SEQUATRE, SECSEPT) sont non-Résidents.

Donc le segment SECSIX, en général présent dans la mémoire centrale, sera évacué sur disque en cas de besoin puis réimplanté dans l'état où il fut évacué. Tandis que les segments SECHUIT ou SEQUATRE plus SECSEPT seront toujours introduits en mémoire dans l'état initial et jamais évacués pour sauvegarder des situations intermédiaires.

REMARQUE 1 : Nombre Priorité, dans la clause SEGMENT-LIMIT, est une borne extérieure aux segments Résidents.

REMARQUE 2 : Si la clause SEGMENT-LIMIT est omise, tous les segments de priorité < 50 sont considérés résidents, ce qui suppose qu'il reste suffisamment de place mémoire disponible pour inclure tour à tour les segments non résidents. Nous conseillons au programmeur d'éviter autant que possible la clause SEGMENT-LIMIT lorsqu'il n'est pas indispensable de sauvegarder les situations intermédiaires de segments car elle implique une écriture sur disque du segment semi-résident avant l'introduction du segment non résident, donc une perte de temps d'évacuation qui peut être assez importante.

En résumé. Pour segmenter un programme il faut :

1. — Découper le programme en Sections indépendantes.

2. — Affecter à chacune de ces sections un numéro de priorité fonction de leur fréquence d'utilisation.

3. — En fonction de la taille de mémoire centrale requise par les différentes sections utiliser la clause SEGMENT-LIMIT pour classer les segments en :

Résidents : Numéro de priorité < Numéro Segment-Limit,
Semi-Résidents : Segment-Limit ⩽ Numéro de priorité < 50.
Non-Résidents : 50 ⩽ Numéro de priorité.

4. — Ne pas oublier le paramètre ØVLY dans les cartes JCL de contrôle du système d'Exploitation pour l'étape de LINK-EDIT du programme.

POINTS PARTICULIERS DE LA SEGMENTATION. RESTRICTIONS

Instruction d'aiguillage ALTER :

— Segments ayant un nombre priorité > 49 : Il est interdit de modifier une instruction GØ TØ d'une section de nombre priorité > 49 si cette section n'est pas présente en mémoire. Autrement dit, une instruction ALTER ne peut affecter une autre section de nombre priorité > 49 que si les deux sections ont même nombre priorité.

En effet, des sections de même nombre priorité constituent un segment et sont toujours présentes ensemble en mémoire centrale. L'instruction GØ TØ « altérée » devra être exécutée avant que le segment ne soit écrasé par un autre car les segments de nombre priorité > 49 ne sont pas sauvegardés mais toujours réintroduits en l'état initial.

— Segments ayant un nombre priorité < 50 : Il est toujours possible d'altérer une section de nombre priorité < 50 sans crainte car ces sections sont ou bien résidentes ou semi-résidentes (donc sauvegardées).

L'ordre ALTER peut être utilisé même si la section affectée n'est pas présente en mémoire au moment de l'exécution de cet ordre.

Instruction PERFØRM : Les restrictions concernant PERFØRM sont semblables à celle de ALTER.

— PERFØRM de Segments ayant un nombre priorité > 49 : On ne peut programmer un PERFØRM de section > 49 que dans le cas où la section contenant le PERFØRM a exactement le même nombre priorité, donc appartient au même segment.

— PERFØRM de Segments ayant un nombre priorité < 50 : Aucune restriction, ce cas est toujours possible.

— Toutefois, en COBOL IBM, l'écriture de PERFØRM est permise dans tous les cas mais il faut savoir qu'à chaque itération PERFØRM d'une section de nombre priorité > 49, la section est fournie dans son état initial indépendamment des itérations précédentes.

Sous-programmes externes appelés par CALL

Ils ne peuvent être segmentés.

CORRIGÉS DES EXERCICES

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 1	GRAPHISME	TYPE DE CARTES	Page 1 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION CRESIGNA (73–80)

```
Page Ligne  A   B
001  010    IDENTIFICATION DIVISION.
     020    PROGRAM-ID.  CRESIGNA.
     030    REMARKS. PROGRAMME DE CREATION PARTIELLE D'UNE BANQUE DE DONNEE
     040        PERSONNEL, FICHIER SIGNALETIQUE ET FICHIER PAIE EN
     050        ORGANISATION DIRECTE ,FICHIER TABLE EN SEQUENTIEL INDEXE.
     060
     070    ENVIRONMENT DIVISION.
     080    INPUT-OUTPUT SECTION.
     090    FILE-CONTROL.
     100        SELECT FIBAND1 ASSIGN TO  UT-S-FIBAND1.
     110        SELECT FIBAND2 ASSIGN TO  UT-S-FIBAND2.
     120        SELECT FITABLE ASSIGN TO  DA-I-FITABLE
     130                       ACCESS IS SEQUENTIAL
     140                       RECORD KEY IS CLETABLE.
     150        SELECT FISIGNAL   ASSIGN TO  DA-D-FISIGNAL
     160                          ACCESS IS SEQUENTIAL
     170                          ACTUAL KEY IS CLESIGNAL
     180                          TRACK-LIMIT IS 800.
     190        SELECT FIPAIE  ASSIGN TO  DA-D-FIPAIE
     200                       ACCESS IS SEQUENTIAL
     210                       ACTUAL KEY IS CLEPAIE
     220                       TRACK-LIMIT IS 1000.
     230    DATA DIVISION.
     240    FILE SECTION.
     250    FD  FIBAND1     RECORDING MODE F    LABEL RECORD STANDARD
     260                    BLOCK CONTAINS 36   RECORDS
     270                    DATA RECORD BAND1.
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME EXERCICE N° 1 — GRAPHISME — TYPE DE CARTES — Page 2 de 4

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION CRESIGNA (73–80)

Page	Ligne	Suite	A	B	
002	010		01	BAND1.	
	020			02 MATR1	PIC 9(6).
	030			02 REST1	PIC X(94).
	040		FD	FIBAND2	RECORDING MODE F LABEL RECORD STANDARD
	050				BLOCK CONTAINS 20 RECORDS
	060				DATA RECORD BAND2.
	070		01	BAND2.	
	080			02 MATR2	PIC 9(6).
	090			02 REST2	PIC X(170).
	100		FD	FITABLE	RECORDING MODE F LABEL RECORD STANDARD
	110				BLOCK CONTAINS 50 RECORDS
	120				DATA RECORD TABLE.
	130		01	TABLE.	
	140			02 FF	PIC XX.
	150			02 CLETABLE	PIC 9(6).
	160			02 PISTES	OCCURS 4 PIC S9(5) COMP.
	170		FD	FISIGNAL	RECORDING MODE F LABEL RECORD STANDARD
	180				DATA RECORD SIGNAL.
	190		01	SIGNAL.	
	200			02 MATR11	PIC 9(6).
	210			02 REST11	PIC X(94).
	220		FD	FIPAIE	RECORDING MODE F LABEL RECORD STANDARD
	230				DATA RECORD PAIE.
	240		01	PAIE.	
	250			02 MATR22	PIC 9(6).
	260			02 REST22	PIC X(170).
	270				

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 1	GRAPHISME	TYPE DE CARTES	Page 3 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION CRESIGNA

Page	Ligne	Suite	A	B
003	010			WØRKING-STØRAGE SECTIØN.
	020		77	F1 PIC X VALUE "1".
	030		77	F2 PIC X VALUE "1".
	040		01	CLESIGNAL.
	050			02 PIST1 PIC S9(5) CØMP VALUE ZERØ.
	060			02 CLEMATR1 PIC 9(8).
	070		01	CLEPAIE.
	080			02 PIST2 PIC S3(5) CØMP VALUE ZERØ.
	090			02 CLEMATR2 PIC 9(8).
	100			PRØCEDURE DIVISIØN.
	110			DEBUT.
	120			ØPEN INPUT FIBAND1 FIBAND2 ØUTPUT FITABLE FISIGNAL FIPAIE.
	130			LEC1.
	140			IF F1 NØT = "1" GØ TØ LEC2.
	150			READ FIBAND1 AT END MØVE 999999 TØ MATRI1.
	160			LEC2.
	170			IF F2 NØT = "1" GØ TØ CØMPAR.
	180			READ FIBAND2 AT END MØVE 999999 TØ MATRI2.
	190			CØMPAR.
	200			MØVE ZERØ TØ PISTES (1) PISTES (2) PISTES (3) PISTES (4)
	210			IF MATRI1 < MATRI2 MØVE "1" TØ F1 MØVE "0" TØ F2 GØ TØ ECRI1.
	220			IF MATRI1 > MATRI2 MØVE "0" TØ F1 MØVE "1" TØ F2 GØ TØ ECRI2.
	230			IF MATRI1 = HIGH-VALUE GØ TØ FINTRA.
	240			MØVE '1' TØ F1 F2 PERFØRM ECRI1 PERFØRM ECRI2.
	250			ECRI3.
	260			MØVE SPACE TØ FF
	270			WRITE TABLE INVALID KEY GØ TØ ERREUR3.

Ref. A 6120

PROGRAMME COBOL

PROGRAMME : EXERCICE N° 1 — GRAPHISME — TYPE DE CARTES — Page 4 de 4

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION : CRESIGNA

Page	Ligne	Suite	A	B
004	010			GØ TØ LEC1.
	020		ECRI1.	
	030			MØVE MATRI1 TØ CLEMATR1 CLETABLE MØVE BAND1 TØ SIGNAL
	040			WRITE SIGNAL INVALID KEY GØ TØ ERREUR1.
	050			MØVE PIST1 TØ PISTES (1).
	060		ECRI11.	
	070			GØ TØ ECRI3.
	080		ECRI2.	
	090			MØVE MATRI2 TØ CLEMATR2 CLETABLE MØVE BAND2 TØ PAIE
	100			WRITE PAIE INVALID KEY GØ TØ ERREUR2.
	110			MØVE PIST2 TØ PISTES (2).
	120		ECRI22.	
	130			GØ TØ ECRI3.
	140		ERREUR1.	
	150			DISPLAY "ECRITURE SIGNALETIQUE IMPØSSIBLE"
	160			DISPLAY BAND1 GØ TØ LEC1.
	170		ERREUR2.	
	180			DISPLAY "ECRITURE PAIE IMPØSSIBLE"
	190			DISPLAY BAND2 GØ TØ LEC1.
	200		ERREUR3.	
	210			DISPLAY "ECRITURE TABLE IMPØSSIBLE"
	220			DISPLAY CLETABLE.
	230		FINTRA.	
	240			CLØSE FIBAND1 FIBAND2 FITABLE FISIGNAL FIPAIE
	250			STØP RUN.
	260			
	270			

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 2	GRAPHISME	TYPE DE CARTES	Page 1 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION TABLUTIL

Page	Ligne	Suite	A	B
001	010		IDENTIFICATIØN DIVISIØN.	
	020		PRØGRAM-ID. TABLUTIL.	
	030		REMARKS. PRØGRAMME UTILISANT LE MØDULE DE GESTIØN DES TABLES.	
	040			
	050		ENVIRØNMENT DIVISIØN.	
	060		INPUT-ØUTPUT SECTIØN.	
	070		FILE-CØNTRØL.	
	080		SELECT FICART ASSIGN TØ UT-S-FICART.	
	090		SELECT FIEDIT ASSIGN TØ UT-S-FIEDIT.	
	100			
	110		DATA DIVISIØN.	
	120		FILE SECTIØN.	
	130		FD FICART LABEL RECØRD ØMITTED	
	140		RECØRDING MØDE F	
	150		DATA RECØRD CARTE.	
	160		01 CARTE.	
	170		02 CØCAR PIC X(3).	
	180		02 DEP PIC 99.	
	190		02 AGE PIC 99.	
	200		02 PRØD PIC 99.	
	210		02 MØNTANT1 PIC 9(7)V99.	
	220		02 MØNTANT2 PIC 9(7)V99.	
	230		02 FILLER PIC X(53).	
	240		FD FIEDIT LABEL RECØRD ØMITTED	
	250		RECØRDING MØDE F	
	260		DATA RECØRD LIGNE.	
	270		01 LIGNE PIC X(133).	

Ref. A 6120

PROGRAMME COBOL

PROGRAMME : EXERCICE N°2 | GRAPHISME | TYPE DE CARTES | Page 2 de 4

PROGRAMMEUR | TÉLÉPHONE | IMPUTATION | DATE | PERFORATIONS | IDENTIFICATION : TABLUTIL

Page	Ligne	Suite	A	B
002	010		WORKING-STORAGE SECTION.	
	020		77 I1 PIC S9(3) COMP.	
	030		77 I2 PIC S9(3) COMP.	
	040		77 I3 PIC S9(3) COMP.	
	050		01 TABLE.	
	060		02 DEPART OCCURS 50 INDEXED BY IND1.	
	070		04 AGENT OCCURS 20 INDEXED BY IND2 IND22.	
	080		06 COL OCCURS 10 INDEXED BY IND3.	
	090		08 COL1 PIC S9(7) V99 COMP.	
	100		08 COL2 PIC S9(7) V99 COMP.	
	110		08 COL3 PIC S9(7) V99 COMP.	
	120		01 LIDET.	
	130		02 FILLER PIC X VALUE SPACE.	
	140		02 DEPL PIC 99.	
	150		02 FILLER PIC X VALUE SPACE.	
	160		02 AGEL PIC 99.	
	170		02 NOMBRE OCCURS 10 INDEXED BY IND33.	
	180		04 NOMB PIC ZZZZZZ9V,99.	
	190		02 FILLER PIC X(26) VALUE SPACE.	
	200			
	210		PROCEDURE DIVISION.	
	220		DEBUT.	
	230		OPEN INPUT FICART OUTPUT FIEDIT.	
	240		PERFORM MIZAZERO VARYING I1 FROM 1 BY 1 UNTIL I1 > 50	
	250		AFTER I2 FROM 1 BY 1 UNTIL I2 > 20	
	260		AFTER I3 FROM 1 BY 1 UNTIL I3 > 10.	
	270		SET IND22 TO 20.	

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 2	GRAPHISME		TYPE DE CARTES	Page 3 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS			IDENTIFICATION TABLUTIL

```
003010 LECTURE.
   020     READ   FICART END GØ TØ CALCULS.
   030     SET  IND1 TØ DEP   SET IND2 TØ AGE    SET IND3 TØ PRØD
   040     ADD MØNTANT1 TØ CØL1 (IND1, IND2, IND3)
   050                     CØL1 (IND1, IND22, IND3)
   060     ADD MØNTANT2 TØ CØL2 (IND1, IND2, IND3)
   070                     CØL2 (IND1, IND22, IND3)
   080     GØ TØ LECTURE.
   090 CALCULS.
   100     PERFØRM CUMUL THRU FINCUM
   110                     VARYING I1 FRØM 1 BY 1 UNTIL I1 > 50
   120                     AFTER   I2 FRØM 1 BY 1 UNTIL I2 > 20.
   130
   140     PERFØRM ECRIT   VARYING I1 FRØM 1 BY 1 UNTIL I1 > 50
   150                     AFTER   I2 FRØM 1 BY 1 UNTIL I2 > 20.
   160     CLØSE FICART FIEDIT    STØP RUN.
   170 MIZAZERØ.
   180     SET IND1 TØ I1  SET IND2 TØ I2  SET IND3 TØ I3
   190     MØVE ZERØ TØ CØL1 (IND1, IND2, IND3)
   200                  CØL2 (IND1, IND2, IND3)
   210                  CØL3 (IND1, IND2, IND3).
   220 CUMUL.
   230     SET IND1 TØ I1      SET IND2 TØ I2
   240     SEARCH CØL AT END GØ TØ FINCUM
   250         WHEN CØL1 (IND1, IND2, IND3) NØT = ZERØ   CØMPUTE
   260         CØL3 (IND1, IND2, IND3) = (CØL2 (IND1, IND2, IND3) *
   270          100) / CØL1 (IND1, IND2, IND3).
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 2	GRAPHISME	TYPE DE CARTES	Page 4 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION TABLUTIL

Page	Ligne	Suite	A	B
004	010		FINCUM.	EXIT.
	020		ECRIT.	
	030			SET IND1 TØ I1 SET IND2 TØ I2
	040			PERFØRM MISENLIGN VARYING I3 FRØM 1 BY 1 UNTIL I3 7 10
	050			MØVE I1 TØ DEPL MØVE I2 TØ AGEL
	060			WRITE LIGNE FRØM LIDET.
	070		MISENLIGN.	
	080			SET IND3 IND33 TØ I3
	090			MØVE CØL3 (IND1, IND2, IND3) TØ NØMB (IND33).
	100			
	110			
	120			
	130			
	140			
	150			
	160			
	170			
	180			
	190			
	200			
	210			
	220			
	230			
	240			
	250			
	260			
	270			

Ref. A 6120

PROGRAMME COBOL

PROGRAMME: EXERCICE N° 3

PROGRAMMEUR:

TÉLÉPHONE | IMPUTATION | DATE | GRAPHISME | PERFORATIONS | TYPE DE CARTES

Page 1 de 4

IDENTIFICATION: TRIINCLU

```
001010 IDENTIFICATION DIVISION.
   020 PROGRAM-ID. TRIINCLU.
   030 REMARKS.  PROGRAMME DE TRI AVEC ELIMINATION ET EDITION
   040                D'ENREGISTREMENTS ERRONES.
   050 ENVIRONMENT DIVISION.
   060 INPUT-OUTPUT SECTION.
   070 FILE-CONTROL.
   080     SELECT FICHENTR    ASSIGN TO UT-S-FICHENTR.
   090     SELECT FICHSORT    ASSIGN TO UT-S-FICHSORT.
   100     SELECT FICHTRIE    ASSIGN TO UT-S-FICHTRIE.
   110     SELECT FICHEDIT    ASSIGN TO UT-S-FICHEDIT.
   120
   130 DATA DIVISION.
   140 FILE SECTION.
   150 SD  FICHTRIE     DATA RECORD TRIE.
   160 01  TRIE.
   170     02  ORIGINE PIC XX.
   180     02  MOIS    PIC XX.
   190     02  NUMORD  PIC X(6).
   200     02  VALIDE.
   210         04  CPTE       PIC  X(6).
   220         04  DIV        PIC  XX.
   230         04  SERV       PIC  X(4).
   240         04  INDIC      PIC  X(6).
   250         04  CONTRE     PIC  X(6).
   260         04  FRANC      PIC  9(8)V99.
   270         04  DOLLAR     PIC  9(8)V99.
```

Réf. A 6120

PROGRAMME COBOL

PROGRAMME EXERCICE N° 3 — GRAPHISME — TYPE DE CARTES — Page 2 de 4

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION TRIINCLU (73–80)

Page	Ligne	Suite	A	B
002	010			04 LIBEL PIC X(26).
	020			02 ZONERR.
	030			04 CODER OCCURS 20 PIC X.
	040		FD	FICHENTR LABEL RECORD STANDARD
	050			BLOCK CONTAINS 3600
	060			RECORDING MODE F
	070			DATA RECORD ENTREE.
	080		01	ENTREE PIC X(100).
	090		FD	FICHSORT LABEL RECORD STANDARD
	100			BLOCK CONTAINS 5600
	110			RECORDING MODE F
	120			DATA RECORD SORTIE.
	130		01	SORTIE PIC X(80).
	140		FD	FICHEDIT LABEL RECORD OMITTED
	150			RECORDING MODE F
	160			DATA RECORD LIGNE.
	170		01	LIGNE PIC X(133).
	180		WORKING-STORAGE	SECTION.
	190		77	I PIC S9(5) COMP.
	200		01	LI1.
	210			02 FILLER PIC X(33) VALUE " OR. MOIS ORDRE. COMPTE".
	220			02 FILLER PIC X(35) VALUE "DIV. SERV INDIC. C. PART".
	230			02 FILLER PIC X(39) VALUE "MONTANT FR. MONTANT DEL".
	240			02 FILLER PIC X(26) VALUE "LIBELLE".
	250		01	LI2.
	260			02 FILLER PIC X VALUE SPACE.
	270			02 ORL PIC XX.

Ref. A 6120

PROGRAMME COBOL

PROGRAMME EXERCICE N° 3

PROGRAMMEUR

TÉLÉPHONE | IMPUTATION | DATE

GRAPHISME

PERFORATIONS

TYPE DE CARTES

Page 3 de 4

IDENTIFICATION TRIINCLU

Page	Ligne	Suite	A	B
003	010			02 FILLER PIC X(4) VALUE SPACE.
	020			02 MML PIC XX.
	030			02 FILLER PIC X(4) VALUE SPACE.
	040			02 ØRDL PIC X(6).
	050			02 FILLER PIC X(4) VALUE SPACE.
	060			02 VALIL.
	070			04 CPTE PIC X(6).
	080			04 FILLER PIC X(4) VALUE SPACE.
	090			04 DIV PIC XX.
	100			04 FILLER PIC X(4) VALUE SPACE.
	110			04 SERV PIC X(4).
	120			04 FILLER PIC X(4) VALUE SPACE.
	130			04 INDIC PIC X(6).
	140			04 FILLER PIC X(4) VALUE SPACE.
	150			04 CØNTRE PIC X(6).
	160			04 FILLER PIC X(4) VALUE SPACE.
	170			04 FRANC PIC 9(8)V,99.
	180			04 FILLER PIC X(4) VALUE SPACE.
	190			04 DØLLAR PIC 9(8)V,99.
	200			04 FILLER PIC X(4) VALUE SPACE.
	210			04 LIBEL PIC X(26).
	220			02 FILLER PIC X(10) VALUE SPACE.
	230		01	LI3.
	240			02 FILLER PIC X(10) VALUE "ERREURS".
	250			02 ZØNE ØCCURS 20.
	260			04 VAL PIC ZZ.
	270			04 VALBLANC PIC XXXX.

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 3	GRAPHISME	TYPE DE CARTES	Page 4 de 4
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION: TRIINCLU

```
004 010        02  FILLER  PIC  X(3)  VALUE SPACE.
    020    PROCEDURE DIVISION.
    030    DEBU1.
    040        OPEN OUTPUT FICHSORT FICHEDIT
    050        PERFORM MISABLANC VARYING I FROM 1 BY 1 UNTIL I > 20
    060        WRITE LIGNE FROM LI1.
    070    DEBU2.
    080        SORT FICHTRIE ASCENDING KEY ORIGINE MOIS NUMORD
    090                             USING FICHENTR
    100                             OUTPUT PROCEDURE EDITION THRU FINED.
    110        CLOSE FICHEDIT FICHSORT    STOP RUN.
    120    EDITION  SECTION.
    130    RETOUR.
    140        RETURN FICHTRIE  AT END GO TO FINED.
    150        IF ZONERR = SPACE GO TO ECRIT.
    160        MOVE ORIGINE TO ORL       MOVE MOIS TO MML
    170        MOVE NUMORD  TO ORDL      MOVE CORRESPONDING VALIDE TO VALIL
    180        WRITE LIGNE FROM LI2
    190        PERFORM METERR VARYING I FROM 1 BY 1 UNTIL I > 20
    200        WRITE LIGNE FROM LI3      GO TO RETOUR.
    210    ECRIT.
    220        WRITE SORTIE FROM TRIE   GO TO RETOUR.
    230    METERR.  IF CODER (I) NOT = "1"  MOVE ZERO TO VAL (I)
    240                           ELSE      MOVE  I   TO VAL (I).
    250    FINED.  EXIT.
    260    MISABLANC.
    270        MOVE  SPACE  TO  VALBLANC (I).
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 4	GRAPHISME	TYPE DE CARTES	Page 1 de 7
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION LISTERRS

```
001  010  IDENTIFICATION DIVISION.
     020  PROGRAM-ID. LISTERRS.
     030  REMARKS. PROGRAMME REPORT WRITER AVEC SECTIONS DECLARATIVES.
     040  ENVIRONMENT DIVISION.
     050  INPUT-OUTPUT SECTION.
     060  FILE-CONTROL.
     070       SELECT FILECT ASSIGN TO UT-S-FILECT.
     080       SELECT FISORT ASSIGN TO UT-S-FISORT.
     090       SELECT FIETAT ASSIGN TO UT-S-FIETAT.
     100  DATA DIVISION.
     110  FILE SECTION.
     120  FD FILECT RECORDING MODE F
     130             LABEL RECORD STANDARD
     140             DATA RECORD ENTREE.
     150  01 ENTREE.
     160       02 COCA PIC XXX.
     170       02 SOCI PIC X(6).
     180       02 DATE PIC X(4).
     190       02 ORIG PIC 99.
     200       02 CPTE PIC X(7).
     210       02 INDI PIC X(6).
     220       02 DEBI PIC 9(8)V99.
     230       02 CRED PIC 9(8)V99.
     240       02 LIBE PIC X(25).
     250       02 CPAR PIC X(7).
     260       02 ZONR.
     270          04 ERRE PIC X OCCURS 20.
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME : EXERCICE N° 4 — GRAPHISME — TYPE DE CARTES — Page 2 de 7

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION : LISTERRS (73 80)

Page	Ligne	Suite	A	B
002	010		FD	FISORT RECORDING MODE F
	020			LABEL RECORD STANDARD
	030			DATA RECORD SORTIE.
	040		01	SORTIE PIC X(100).
	050		FD	FIETAT RECORDING MODE F
	060			LABEL RECORD STANDARD
	070			REPORT ETAT.
	080		WORKING-STORAGE SECTION.	
	090		77	DEBCAL PIC S9(8)V99 COMP-3 VALUE ZERO.
	100		77	CRECAL PIC S9(8)V99 COMP-3 VALUE ZERO.
	110		77	EDDEB PIC S9(8)V99 COMP-3 VALUE ZERO.
	120		77	EDCRE PIC S9(8)V99 COMP-3 VALUE ZERO.
	130		77	IND PIC S9(4) COMP.
	140		01	CARTE.
	150		02	CARORG PIC 99.
	160		02	CARDEB PIC 9(8)V99.
	170		02	CARCRE PIC 9(8)V99.
	180		02	FILLER PIC X(58).
	190		01	TABLE.
	200		02	ELEM OCCURS 98.
	210		04	DEBTAB PIC S9(8)V99 COMP-3.
	220		04	CRETAB PIC S9(8)V99 COMP-3.
	230		01	TABERR.
	240		02	ELEM OCCURS 20.
	250		04	NUMER PIC ZZ.
	260		04	BLANC PIC XX.
	270			

Ref. A 6120

PROGRAMME COBOL

PROGRAMME	EXERCICE N° 4	GRAPHISME	TYPE DE CARTES	Page 3 de 7
PROGRAMMEUR	TÉLÉPHONE / IMPUTATION / DATE	PERFORATIONS		IDENTIFICATION LISTERRS

Page	Ligne	Suite	A	B
003	010		REPØRT	SECTIØN.
	020		RD	ETAT CØNTRØL ØRIG
	030			PAGE LIMIT 60 LINE HEADING 1
	040			FIRST DETAIL 5 LAST DETAIL 55
	050			FØØTING 56.
	060		01	TETETAT TYPE REPØRT HEADING NEXT GRØUP NEXT PAGE.
	070			02 LINE 1.
	080			04 CØLUMN 24 PIC X(39) VALUE
	090			"LISTE DES ANØMALIES DE CØMPTABILITE".
	100			04 CØLUMN 93 PIC X(4) VALUE "PAGE".
	110			04 CØLUMN 99 PIC ZZZ9 SØURCE PAGE-CØUNTER.
	120			02 LINE 5.
	130			04 CØLUMN 1 PIC X(50) VALUE "01 ERREUR CØDE CARTE".
	140			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	150			"02 ERREUR CØDE SØCIETE".
	160			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE "03 ERREUR DATE".
	170			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	180			"04 ØRIGINE INVALIDE".
	190			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	200			"05 CØMPTE INVALIDE".
	210			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	220			"06 INDICATIF INVALIDE".
	230			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	240			"07 MØNTANT DEBIT ERRØNE".
	250			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE
	260			"08 MØNTANT CREDIT ERRØNE"
	270			02 LINE PLUS 1 CØLUMN 1 PIC X(50) VALUE

Ref. A 6120

PROGRAMME COBOL

PROGRAMME: EXERCICE N° 4

PROGRAMMEUR:

TÉLÉPHONE	IMPUTATION	DATE	GRAPHISME	PERFORATIONS	TYPE DE CARTES	Page 4 de 7	IDENTIFICATION
							LISTERRS

```
004010          "09 CONTRE-PARTIE INVALIDE".
   020     02 LINE PLUS 1 COLUMN 1 PIC X(50) VALUE
   030          "10 INCOMPATIBILITE SOCIETE*ORIGINE*COMPTE".
   040     02 LINE PLUS 1 COLUMN 1 PIC X(50) VALUE
   050          "11 INCOMPATIBILITE SOCIETE*ORIGINE*CONTRE-PARTIE".
   060 01  TETPAGE TYPE PAGE HEADING.
   070     02 LINE 1.
   080        04 COLUMN 24 PIC X(39) VALUE
   090           "LISTE DES ANOMALIES DE COMPTABILITE".
   100        04 COLUMN 93 PIC X(4) VALUE "PAGE".
   110        04 COLUMN 99 PIC ZZZ9 SOURCE PAGE-COUNTER.
   120 01  LIDETAIL TYPE DETAIL.
   130     02 LINE PLUS 2.
   140        04 COLUMN 1 PIC X(43) VALUE
   150           "CC  SOCIETE  DATE  OR  COMPTE    INDIC.".
   160        04 COLUMN 49 PIC X(19) VALUE "DEBIT        CREDIT".
   170        04 COLUMN 82 PIC X(7) VALUE "LIBELLE".
   180        04 COLUMN 101 PIC X(7) VALUE "C.PART.".
   190     02 LINE PLUS 1.
   200        04 COLUMN    1 PIC  XX      SOURCE COLA.
   210        04 COLUMN    7 PIC  X(6)    SOURCE SOCI.
   220        04 COLUMN   16 PIC  X(4)    SOURCE DATE.
   230        04 COLUMN   23 PIC  99      SOURCE ORIG.
   240        04 COLUMN   28 PIC  X(7)    SOURCE CPTE.
   250        04 COLUMN   38 PIC  X(6)    SOURCE INDI.
   260        04 COLUMN   47 PIC 9(8)V99  SOURCE DEBI.
   270        04 COLUMN   60 PIC 9(8)V99  SOURCE CRED.
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME EXERCICE N° 4 — GRAPHISME — TYPE DE CARTES — Page 5 de 7

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION LISTERRS (73–80)

Page	Ligne	Suite	A	B	
005	010			04	COLUMN 73 PIC X(25) SOURCE LIBE.
	020			04	COLUMN 101 PIC X(7) SOURCE CPAR.
	030			02	LINE PLUS 1.
	040			04	COLUMN 1 PIC X(7) VALUE "ERREURS".
	050			04	COLUMN 11 PIC X(80) SOURCE TABERR.
	060		01	ERORIGINE TYPE CONTROL FOOTING ORIG NEXT GROUP NEXT PAGE.
	070			02	LINE PLUS 2.
	080			04	COLUMN 33 PIC X(9) VALUE "THEORIQUE".
	090			04	COLUMN 45 PIC ZZBZZZBZZ9V,99 SOURCE DEBTAB (ORIG).
	100			04	COLUMN 74 PIC X(9) VALUE "THEORIQUE".
	110			04	COLUMN 85 PIC ZZBZZZBZZ9V,99 SOURCE CRETAB (ORIG).
	120			02	LINE PLUS 1.
	130			04	COLUMN 1 PIC X(7) VALUE "ORIGINE".
	140			04	COLUMN 10 PIC XX SOURCE ORIG.
	150			04	COLUMN 14 PIC X(45) VALUE
	160				"ERRONEE DEBIT ********************".
	170			04	COLUMN 66 PIC X(33) VALUE
	180				"CREDIT ********************".
	190			02	LINE PLUS 1.
	200			04	COLUMN 33 PIC X(7) VALUE "CALCULE".
	210			04	COLUMN 45 PIC ZZBZZZBZZ9V,99 SOURCE EDDEB.
	220			04	COLUMN 74 PIC X(7) VALUE "CALCULE".
	230			04	COLUMN 85 PIC ZZBZZZBZZ9V,99 SOURCE EDCRE.
	240		01	DERPAGE TYPE REPORT FOOTING LINE NEXT PAGE.
	250			02	LINE 10 COLUMN 30 PIC X(23) VALUE
	260				"FIN DE LISTE ERREURS".
	270				

Ref. A 6120

PROGRAMME COBOL

PROGRAMME: EXERCICE N° 4

PROGRAMMEUR:

TÉLÉPHONE | IMPUTATION | DATE | GRAPHISME | PERFORATIONS | TYPE DE CARTES

Page 6 de 7

IDENTIFICATION: LISTERRS

```
006 010 PRØCEDURE DIVISIØN.
    020 DECLARATIVES.
    030 SECØRIGIN SECTIØN. USE BEFØRE REPØRTING ERØRIGINE.
    040 DEBSEC.
    050     IF ERRE (4) = "1" MØVE 1 TØ PRINT-SWITCH  GØ TØ SECZERØ.
    060     IF DEBCAL NØT = DEBTAB (ØRIG)  GØ TØ SEC1.
    070     IF CRECAL NØT = CRETAB (ØRIG)  GØ TØ SEC1.
    080     MØVE 1 TØ PRINT-SWITCH  GØ TØ SECZERØ.
    090 SEC1.
    100     MØVE ZERØ TØ PRINT-SWITCH
    110     MØVE DEBCAL TØ EDDEB   MØVE CRECAL TØ EDCRE.
    120 SECZERØ.
    130     MØVE ZERØ TØ DEBCAL CRECAL.
    140 FINSEC. EXIT.
    150 END DECLARATIVES.
    160 DEBUT.
    170     ØPEN INPUT FILECT  ØUTPUT FISØRT FIETAT.
    180     PERFØRM MISABLAN VARYING IND FRØM 1 BY 1 UNTIL IND 7 20.
    190 LEC1.
    200     ACCEPT CARTE   IF CARØRG = 99 GØ TØ LEC2.
    210     MØVE CARDEB TØ DEBTAB (IND)
    220     MØVE CARCRE TØ CRETAB (IND)   GØ TØ LEC1.
    230 LEC2.
    240     READ FILECT AT END DISPLAY "FICHIER VIDE" GØ TØ FINTRA.
    250     INITIATE ETAT.
    260 TRAIT.
    270     IF ZØNR = SPACE GØ TØ ECRISØR.
```

Ref. A 6120

PROGRAMME COBOL

PROGRAMME : EXERCICE N° 4 — GRAPHISME — TYPE DE CARTES — Page 7 de 7

PROGRAMMEUR — TÉLÉPHONE — IMPUTATION — DATE — PERFORATIONS — IDENTIFICATION : LISTERRS (73 80)

Page	Ligne	Suite	A	B
007	0110			PERFORM MECODER VARYING IND FROM 1 BY 1 UNTIL IND > 20.
	0210			GENERATE LIDETAIL.
	0310		LECTURE.	
	0410			READ FILECT AT END GO TO FINDETA.
	0510			GO TO TRAIT.
	0610		FINDETA.	
	0710			TERMINATE ETAT.
	0810		FINTRA.	
	0910			CLOSE FILECT FIETAT FISORT
	1010			STOP RUN.
	1110		ECRISOR.	
	1210			WRITE SORTIE FROM ENTREE
	1310			ADD DEBI TO DEBCAL ADD CRED TO CRECAL
	1410			GO TO LECTURE.
	1510		MIGABLAN.	
	1610			MOVE SPACE TO BLANC (IND).
	1710		MECODER.	
	1810			IF ERRE (IND) = "1" MOVE IND TO NUMER (IND)
	1910			ELSE MOVE ZERO TO NUMER (IND).
	2010			
	2110			
	2210			
	2310			
	2410			
	2510			
	2610			
	2710			

Ref. A 6120

INDEX ALPHABÉTIQUE

ACCESS
Organisation Directe, 7.
— Relative, 25.
— Séquentiel Indexé, 40.
— VSAM consecutive, 76.
— VSAM indexée, 84.
— VSAM Relative 97.
ACTUAL KEY
Organisation Directe, 4-5, 7-8, 12, 14, 18.
ALTERNATE AREA (RESERVE)
Organisation Sequentiel Indexé, 40.
ALTERNATEINDEX, 110-113.
ALTERNATE KEY, 115, 117.
A. M. S. Langage VSAM, 53.
APPLY CØRE-INDEX
Organisation Sequentiel Indexé, 42.
CYL-ØVERFLØW
Organisation Sequentiel Indexé, 42.
REØRG-CRITERIA
Organisation Sequentiel Indexé, 42.

BLDINDEX Commande VSAM, 112, 118.

CATALOGUE VSAM, 56 à 60.
CF Report Writer, 158-159.
CH Report Writer, 158-159.
CLØSE
Organisation Directe, 10.
— Relative, 27.
— Sequentiel Indexé, 44.
— VSAM Consecutive, 77.
— VSAM Indexée, 87.
— VSAM Relative, 100.
CLUSTER Organisation VSAM, 56, 61 à 64.
Index Secondaire, 106.
CØDE Report Writer, 153.
CØLUMN NUMBER Report Writer, 160.
CONSÉCUTIVE VSAM (ORGANISATION), 73 à 79.
CØNTRØL AREA Organisation VSAM, 54-55.
INTERVAL Organisation VSAM, 54-55, 62-64.
Report Writer, 154.
FØØTING Report Writer, 158, 159, 163.
HEADING Report Writer, 158, 159, 163.
CØRE-INDEX
Organisation Sequentiel Indexé, 42.
CREATION
Langage AMS-VSAM, 56 à 60.
Organisation Directe Sequentielle, 12 à 17.
— — Random, 18-19.
— Relative Sequentielle, 28-29.
— Sequentiel Indexé-Sequentielle, 45 à 47.
— VSAM-Consécutive, 78-79.
— VSAM-Indexée, 87 à 89.
— VSAM Relative, 101-102.
— Index Secondaire, 118.

DE Report Writer, 158-159.
DECLARATIVES
Organisation VSAM Consecutive, 77.
— — Indexée, 86.
Report Writer, 158-159.
DELETE
Organisation VSAM Indexée, 87.
— VSAM Relative, 100.
DETAIL Report Writer, 155, 158-159, 163.
DIRECTE (ORGANISATION), 3, 22.

EXTENDED-SEARCH
Organisation Directe, 9.

FILE STATUS, 71, 72.
Organisation VSAM Consecutive, 76.
— — Indexée, 84-85.
— — Relative, 97.

FIRST DETAIL Report Writer, 155.
FØØTING Report Writer, 155, 158-159, 173.

GENERATE Report Writer, 165.
GIVING Tri, 143.
GRØUP INDICATE Report Writer, 160.
GROUPES D'EDITION Report Writer, 156 à 164.

HEADING Report Writer, 155, 158-159, 163.

INDEX
  Organisation Sequentiel Indexé, 33 à 38.
  Tables, 128-131.
  (USAGE IS...), 131.
INDEXED BY Tables, 129-130.
INDEXEE (ORGANISATION VSAM), 80 à 93.
INITIATE Report Writer, 164.
INPUT PRØCEDURE Tri, 132, 144.
INVALID KEY
  Organisation Directe, 9, 10, 20.
  — Relative, 26.
  — Sequentiel Indexé, 44.
  — VSAM Indexée, 85-87.
  — VSAM Relative, 98-100.

KEY Index Secondaires, 115-117.

LAST DETAIL Report Writer, 155.
LINE NUMBER Report Writer, 157.
LINE-CØUNTER Report Writer, 163.
LØCK (CLØSE WITH...)
  Organisation Directe, 10.
  — Relative, 27.
  — Sequentiel Indexé, 44.
  — VSAM Consecutive, 77.
  — VSAM Indexée, 87.
  — VSAM Relative, 100.

MERGE Tri, 149.
MISE A JOUR
  Organisation Directe, 20 à 22.
  — Relative, 30 à 32.
  — Sequentiel Indexé, 47 à 50.
  — VSAM Indexée, 91 à 93.
  — VSAM Relative, 103 à 105.
  — Index Secondaire, 118 à 121.

NEXT GRØUP Report Writer, 157, 163.
  PAGE Report Writer, 158, 163.
NØMINAL KEY
  Organisation Relative, 24, 26, 28.
  — Sequentiel Indexé, 33, 41.

ØCCURS Tables, 128.
ØRGANIZATIØN
  Organisation VSAM consecutive, 76.
  — — Indexée, 84.
ØPEN
  Organisation Directe, 8.
  — Relative, 26.
  — Sequentiel Indexé, 43.
  — VSAM Consecutive, 77.
  — VSAM Indexée, 85.
  — VSAM Relative, 98.
ØUTPUT PRØCEDURE Tri, 139, 144.
ØVERFLØW
  Organisation Sequentiel Indexé, 35 à 38.

PAGE FØØTING Report Writer, 158, 163.
  HEADING Report Writer, 158, 163.
  LIMIT Report Writer, 155.
PAGE-CØUNTER Report Writer, 163.
PASSWØRD
  Langage AMS-VSAM, 58-59.
  Organisation VSAM Consecutive, 76.
  — VSAM Indexée, 84.
  — VSAM Relative, 97.
PF Report Writer, 158, 163.
PH Report Writer, 158, 163.
PRINT-SWITCH Report Writer, 164, 167.

RANDØM (ACCESS...)
  Organisation Directe, 7, 18, 20.
  — Relative, 25, 30.
  — Sequentiel Indexé, 40, 47.
  — VSAM Indexée, 84.
READ
  Organisation Directe, 9.
  — Relative, 26.
  — Sequentiel Indexé, 44.
  — VSAM Consecutive, 77.
  — VSAM Indexée, 85.
  — VSAM Relative, 99.
RECØRD KEY
  Organisation Sequentiel Indexé, 33, 40.
  — VSAM Indexée, 84.
RECØRD-ØVERFLØW
  Organisation Directe, 8.
  RELATIVE KEY, 97.
RELEASE Tri, 145.
REPØRT FØØTING Report Writer, 158, 162.

HEADING Report Writer, 158-162.
SECTIØN Report Writer, 153-163.
RERUN Tri, 142.
Organisation VSAM Consecutive, 76.
— VSAM Indexée, 85.
RESERVE ... ALTERNATE AREA
Organisation Sequentiel Indexé, 40.
RESET ØN Report Writer, 162.
RETURN Tri, 145.
REWRITE
Organisation Directe, 6, 10, 20.
— Relative, 26.
— Sequentiel Indexé, 44.
— VSAM Consecutive, 77.
— VSAM Indexée, 86.
— VSAM Relative, 100.
RF Report Writer, 158.
RH Report Writer, 158.

SAME RECØRD AREA
Tri, 142.
Organisation VSAM, 74.
SD Tri, 142.
SEARCH Tables, 132 à 137.
VARYING Tables, 133 à 135.
ALL Tables, 135 à 137.
SEEK
Organisation Directe, 9.
SEGMENTS Segmentation, 178-179.
SELECT
Organisation Directe, 6.
— Relative, 25.
— Sequentiel Indexé, 39.
— Consecutive VSAM, 75.
— Indexée VSAM, 83.
— Relative VSAM, 96.
Report Writer, 152.
Tri, 141.
SEQUENTIAL (ACCESS MØDE IS...)
Organisation Directe, 7.
— Relative, 25.
— Sequentiel Indexé, 40.
— VSAM Consecutive, 76.
— VSAM Indexée, 84.
— VSAM Relative, 97.
SET Tables, 127, 130.
SØRT Tri, 138 à 149.
SØRT-CØRE-SIZE Tri, 146.
SØRT-FILE-SIZE Tri, 146.
SØRT-MØDE-SIZE Tri, 146.
SØRT-RETURN Tri, 146.
SØURCE Report Writer, 161.
START
Organisation Sequentiel Indexé, 43-44.
— VSAM Indexée, 85.
— VSAM Relative, 98.
STATUS (FILE)
Organisation VSAM Consecutive, 76.
— VSAM Indexée, 84.
— VSAM Relative, 97.
SUM Report Writer, 161.

TERMINATE Report Writer, 166.
TRACK-AREA
Organisation Sequentiel Indexé, 41.
TRACK-LIMIT
Organisation Relative, 8.
TYPE IS Report Writer, 158.

UNIT (CLØSE...)
Organisation Directe, 10.
— Relative, 27.
UPDATE
Organisation Directe, 20 à 22.
— Relative, 30 à 32.
— Sequentiel Indexé, 47 à 50.
USE AFTER STANDARD-Relative VSAM, 99.
USE BEFØRE REPØRTING Report Writer, 166.
USING Tri, 138-139, 143.

VSAM Langage AMS, 53 à 72.
Organisation Consecutive, 73 à 79.
— Indexée, 80 à 93.
— Relative, 94 à 105.
Index Secondaires, 106 à 125.

WITH CØDE Report Writer, 153.
WRITE
Organisation Directe, 6, 7, 10.
— Relative, 26.
— Sequentiel Indexé, 44.
— VSAM Consecutive, 77.
— VSAM Indexée, 86.
— VSAM Relative, 99.

Imprimerie de la Manutention à Mayenne
Dépôt légal : Janvier 1986
N° d'éditeur : 4455

Avec ses nombreux exemples et la fiabilité de ses informations, ce livre vous permet de pénétrer les secrets des instructions les plus évolués du langage Cobol.

Il comprend quatre parties :

La première concerne Cobol et les différents types d'organisations de fichiers (directe, relative et séquentielle indexée).

La seconde partie décrit le nouveau module de gestion des tables qui permet l'indexage direct ou indirect et comprend des verbes inédits pour leur gestion.

La troisième partie, concerne, quant à elle, les modules d'édition et de tri Cobol.

Quant à la dernière partie, elle comprend un recueil fort utile de toutes les différences qui existent entre un Cobol quelconque et un Cobol ANS.

EYROLLES

8565

EYROLLES

LES EXTENSIONS AU COBOL A.N.S.

Ch. BONNIN